National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Service des thèses canadiennes

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

UNIVERSITY OF ALBERTA

Natural Language Explanation of Natural Deduction Proofs

BY

Andrew Edgar

A thesis submitted to the Faculty of Graduate Studies and Research in partial fullfillment of the requirements for the degree of Master of Science.

DEPARTMENT OF COMPUTING SCIENCE

Edmonton, Alberta
Fall 1991

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-70085-8

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Andrew Edgar

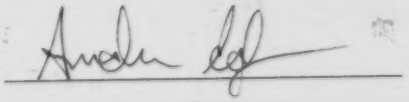
TITLE OF THESIS: Natural Language Explanation of Natural Deduction Proofs

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1991

Permisson is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



603-10125 109th Street

Edmonton, Alberta

T5J 3P1

Oct. 10/91

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recomended to the Faculty of Graduate Studies and Research for acceptance a thesis entitled Natural Language Explanation of Natural Deduction Proofs submitted by Andrew Edgar in partial fulfillment of the requirements for the degree of Master of Science.

F.J. Pelletier

Randy Goebel

Paul Sorenson

Bernard Linsky

Oct. 8/91

ABSTRACT

Automated theorem proving has been claimed to have a wide variety of uses, from program verification and generation, to robotic planning systems, and as an explanation of various cognitive abilities. But for many of these applications, it would be beneficial if the user could inspect the background proof that gave rise to the course of action or the answer. But, if the user is not a logician, it is not helpful for the system to merely display the proof. Rather it should be put in a form understandable by the user; and an obvious solution would be a natural language "back end" which could explain the proof in ordinary language. The goal is to produce an explanation facility for the natural deduction theorem prover THINKER. This program should be able to generate a wide variety of explanations for any individual proof.

The program EXPLAIN was written to produce the explanations from the proofs generated by THINKER. There are four distinct dimensions along which they can be varied. (i) The method used to produce the explanation: top-down, bottom-up, or a mixture of the two. (ii) The level of the explanation: ranging from complete to a high-level overview. (iii) The amount of explanation done for the lines in the proof: explaining just the inference rules, inference rules plus the connectives, or complete explanation. (iv) When the explanation is done: either statically (after the fact) or dynamically (during the proof generation process).

Acknowledgement

I would like to thank my supervisor Jeff Pelletier and co-supervisor Randy Goebel for all the help that they gave to me. Thank you also goes to Abdul, Aditya, Shurjo, Ron K., Robert, Charles, Dekang, and everyone else from the AI group. Ron H. helped by putting up with me and being a fellow Winnipeg fan while stuck in the Edmonton wastelands. Finally, I would like to thank Mom, Dad, Stuart and Aaron for all the support and encouragement that they gave me over the years.

Table of Contents

1. Introduction and Natural Deduction 1

1.1 Uses for Automated Theorem Proving 1

1.2. Explanations for Proofs in Applications 4

1.3. Natural Deduction Theorem Proving 7

1.4. Multiple Explanations of Proofs 16

2. Background of Explaining Natural Deduction Proofs 19

2.1. Introduction 19

2.2. Chester's Paper 19

2.3. McDonald's Paper 26

2.4. Felty and Hager's Paper 29

2.5. Huang's Paper 32

3. How to Generate Many Explanations 39

3.1. Explaining Proofs: Dynamics, Formulas, Levels, and Methods 39

3.2. Explanations of the Formulas 44

3.2.1. Explanation of the Inference Rules 44

3.2.2. Explanation of the Connectives 47

3.2.3. Explanation of the Predicates 50

3.3. Levels of Explanation 53

3.4. Methods of Explanation 54

3.4.1. Bottom-up Method 54

3.4.2. Top-down Method 56

3.4.3. Sample Proof and the Order of Explanation 57

3.4.4. Mixed Method 58

4. Examples 59

4.1. Examples of Proof Explanations 60

5. Future Developments and Conclusions 93

5.1. Areas for Future Work 93

5.2. Conclusions 95

5.3. Portability 99

Bibliography 100

List of Figures

Figure 1: First graph of example proof 21

Figure 2: Final Graph showing lines grouped into paragraphs 23

Figure 3: Example proof 1 45

Figure 4: Example proof 2 57

1. Introduction and Natural Deduction

1.1 Uses for Automated Theorem Proving

Automated theorem proving has classically been viewed as having a wide range of applications: from program verification and generation, through validating the design of circuits. It has also been used for designing superior logic circuits and in question/answer systems. Other applications include robotic planning systems, the answering of open questions in mathematics and in logic, and man-machine cooperative systems. Finally the investigation of human cognition and natural language understanding systems are also done using automated theorem proving.

One of the first uses of automated theorem proving was for programs that verify the correctness of computer programs [Boyer84a, Boyer84b, Good82]. The automatic theorem prover would mechanically prove that a given program satisfies some specification, or produces the same output as some other program, or can be executed within certain time and space bounds. To do this, there must be a formal program semantics given for the programming language, and an automated program verifier reduces the question of whether the program has a particular property to the question of whether certain formulas are theorems which respect the formal semantics. Along these same lines, one might wish to present a formal statement of an algorithm and have an automated theorem prover generate explicit code in some programming language; that is, develop an automatic programmer [Bibel79]. Or even more ambitiously, one might wish to only specify the relations that are to hold between the input and output variables, and to have an automated theorem prover try to generate a proof of $(\forall x(Px \rightarrow (\exists y)Qxy)$ where x ranges

over input variables to the program and P is the predicate that the input is expected to satisfy, y ranges over output variables and Q specifies the relation between each input variable and its associated output variable after execution of the desired program. Given enough other axioms so that this formula is provable, the proof of the formula can be converted into a program by rather well-understood mechanisms. This is program generation.

A method similar to that used in the verification of programs can be used to validate the design of circuits [Hunt86]. We are given an already-designed circuit and the specifications that it supposedly satisfies, and the goal is to prove the design actually does meet these specifications. This type of validation can be thought of as language translation from the language representing how the circuit is built to one that represents the correct output of the circuit. Automated theorem proving may also be used to design superior logic circuits [Wojciechowski83]. In this work the automatic reasoner will be given the specifications for a circuit and asked to design one that satisfies all the specifications. An automated reasoning program ITP [Lusk82a, Lusk82b] has also been used to verify the software and hardware fault-tolerant properties of a system designed to monitor a nuclear reactor [Chisholm85].

Another group of uses to which automated theorem proving is put contains question/answer systems. Not only can theorem provers provide answers to yes/no questions to a database, but long ago it was shown how to use theorem proving techniques to extract answers from a declarative database. And a slight generalization of the method can be used to answer "how" questions that require the specification of a sequence of actions to perform a task. This suggested how to use automated reasoning to automatically specify a "plan", and this [STRIPS/ABSTRIPS] robotic planning methodology is seen to apply widely to all types of mechanical generation of a sequence of actions that

will accomplish a goal which was stated at a high level of generality. The theorem prover begins with the initial state of the world and proves by the use of actions that the final goal state can be achieved, where performing an action will change the state of the world. The plan would then be the sequence of rules, or actions, in the proof used to move from one state of the world to the next.

Finally there are the more "cognitive science" applications of automated theorem proving. One such application area is in mathematics and formal logic. Sseveral previously open questions in mathematics have been answered with the assistance of automated reasoning [Winker79, Winker81]. Some previous open questions were answered by the program supplying a proof, some by it generating a finite model, and others by the program generating a counterexample. These methods were used in solving open questions in the field of ternary boolean algebra [Wos82]. A ternary boolean algebra is a nonempty set satisfying the following five axioms:

1) f(f(v,w,x),y,f(v,w,x)) = f(v,w,f(x,y,z))

2) f(y,x,x) = x

3) f(x,y,g(y)) = x

4) f(x,x,y) = x

5) f(g(y),y,x) = x

where the function, f, acts as a so-called "product" and the function, g, acts as a so-called "inverse". The open question is to determine which (if any) of these five axioms is independent of the remaining set of four. Axioms 4 and 5 have been known to be dependent. Dependencies can be established by the standard use of a theorem prover, that being finding a proof. To determine independence the theorem prover was used in a different way. It was asked to form models that satisfied various sets of four of the

axioms but failed to satisfy the fifth. These three models (one not satisfying axiom 1,2 and 3) were very small, each consisting of but three elements. By this method of generating models and counterexamples it was found that each of the three axioms were independent.

An automated theorem prover can also be used in the field of formal logic to answer open questions [Wos84b]. The idea is that a person will guide the computer, making overall strategic suggestions and allow the program to organize and prove lower level portions of the proofs. The automated reasoner will act as a collegue and be asked to help with the formulation, testing, proof, and refutation of conjectures. In the realm of language understanding (by humans), it has been claimed that people engage in (subconscious) theorem proving to draw inferences from "what is said" to "what is meant"; that they must be able to draw these inferences (again subconsciously) on a wide variety of topics to be able to know (for example) what events occur before which other events in a tale that is narrated to them. They also draw a wide variety of inferences based on "world knowledge" together with what is said to be able just to minimally understand what the speaker intends. Therefore, any computer natural language understanding system must be able to invoke a theorem prover in order merely to attain minimal competence. Furthermore, since rational reasoning has long been seen as what sets people apart from other animals, investigations into computerized theorem proving has been sometimes thought to be relevant to understanding human psychology.

1.2 Explanations of Proofs in Applications

The previously-mentioned uses of automated theorem proving would be a great boon if they were successfully completed; however they are currently only marginally acceptable. Even if a scientific breakthrough in the field were to happen, it is surely the

case that the general population (or indeed, even the scientific community) would justifiably be wary of any such claim, and would demand some ability to investigate whether any particular use was correct. We are all familiar with such challenges as "Would you trust a never-tested train routing program if it were automatically verified?" or "Do you believe this nuclear power plant to be safe on the basis of an automated reasoner's proof that its circuitry has the predicate 'fail-safe' truly applied to it?" And we all recall the reaction of the computer science community to the claim that the space lasers of the Strategic Defense Initiative would be guided by a program too large to be comprehended by a person but that could be automatically verified.

But we needn't go to such extremes to find cases where we would like the option of having humans be able to inspect and understand the proofs generated by an automated theorem prover. In the setting of a mathematician interacting with an automated theorem prover, it is certain that when an automated reasoner has produced a proof to an open question an explanation of the proof would be desired to aid in its understanding. Even when the program has not come up with a proof examination of the work done by the system could be an aid in developing new conjectures. In the area of program generation, if a theorem prover alleges that its program is able to perform some task, it would be the height of folly not to inspect the background proof. One certainly wants to be able to know how a question/answer system arrived at its answer, especially in those cases where the system used a lot of theorem proving power to generate it. And if we are truly interested in people's cognitive abilities, we are not merely interested in the assertion that a theorem prover might make to the effect that a certain formula is a theorem, but rather we are interested in how the formula was proved to be a theorem and whether this corresponds to how people do similar proofs.

When a natural language understanding program is employed to represent the "underlying meaning" of a story or an text, we certainly would like to have the ability to inspect for ourselves whether we think the system has hit on the correct understanding of the text as we know it. An example of this would be for a system that takes natural language as input and then uses this to perform some actions. It would be helpful to examine the proof which did the interpretation of the input to see if there was any misinterpretation and thus avoid any incorrect actions. Finally, in a robotic planning system we would like to inspect the plan before it is executed to check for any unseen side-effects or in flaws in the reasoning. A extreme example could be a system in a space ship that would react to any emergency such as a fire. The system could reason that a fire can not exist without oxygen and thus it will open the hatch and release the air from the ship. The flaw is, of course, that the humans on the ship will perish without the air. This type of extreme flaw in reasoning could occur and thus the examination of a system's plan before its execution would be advantageous.

What is the best way for us to inspect the robotic plan (or any other application of automated theorem provers)? This would depend on the particular area under investigation and the expertise of the inspector. But for the types of examples mentioned above we would like to be able to examine the background proof that was generated by the automated theorem prover. We would like to be able to see whether it really does prove the hidden sublemma, or if the robotic plan holds undesirable side-effects, and we might be curious to how the system provided the answer to our query.

For many of the uses of automated theorem proving, the simple inspection of the generated background proof is insufficient to the understanding. A printout of the proof may be useful to some logician or to the creator of the system; but what is desired is that the immediate user of the system be able to intuitively understand the reasoning or logic

behind the answer. And given that these users are unlikely to be formal logicians with a computer background, a detailed formal proof would be of little use. Rather, we would like something which could capture the reasoning involved in the application but that presents the information to the user at his/her own level of understanding and in a familiar language. The most plausible candidate for a representation of the proof (which is being used by the system) would be one presented in natural language. This would obviate the need for any special understanding of the problem by the user. Of course, the user must still be able to follow the presentation of the proof; but we presume that some things would be much more striking when presented in natural language.

It therefore seems that the ability to present a proof that was generated by an automated theorem prover in a natural language representation would be useful. Or at least, having this ability would alleviate many of the qualms that people have in trusting automated theorem proof generation. We call this ability of a computational system to "explain itself" or to "justify itself" by the use of a natural language representation of the internal proof generated and used, a *natural language back end*.

The explanation of the proof could used in one of two ways. First, the natural language presentation could be used as the only representation given to the user. In this way the explanation must be very clear and have no ambiguities that would make it difficult to understand. Secondly, the proof can also be given to the user and the natural language explanation would act as a guide to aid him/her in understanding the proof. The explanation need not be as exacting with this type of presentation, since the explanation could refer to the proof and be a guide to its understanding.

1.3 Natural Deduction Theorem Proving

Although resolution-based inference is perhaps the industry standard in automated theorem proving, there have always been systems that employed a different format. Even in the late 50's and early 60's there were different systems: The Logic Theorist produced proofs using an axiomatic method, and the output of the program would be considered legitimate axiomatic proofs; Wang's systems employed a Gentzen-sequent proof strategy; Beth's systems employed his semantic tableaux method; and Prawitz's systems seem to use a natural deduction format.

There are many differences between a natural deduction style of logic and a resolution style. One obvious difference is that natural deduction does not convert formulas to any normal form (e.g., negated-conclusion clause form) but instead works with them in their original, "natural form" [Murray82]. This is not such an important difference, since resolution-like strategies can and have been developed which also do resolution on "natural form" formulas. (And in any case, sequent proof strategies and semantic tableaux methods both operate on "natural form" formulas). More important is the method of developing a proof in natural deduction systems. The fundamental idea is that for each type of ("natural") connective there are two ways to operate with a formula that has it as a main connective: if such a formula is already in the proof one "breaks it down" to get the component parts of the formula, and if it is not in the proof but (for whatever reason) we desire it to be in the proof there is a method for introducing the formula.

In any of the various natural deduction systems, some of the above-mentioned ways to operate with a formula amount to saying: "if formulas Φ_1, Φ_2,.... are already in the proof, then we are entitled to add formula Ψ to the proof". Such operations are called *rules of inference*. More strikingly, and the feature that many think of as defining natural deduction, is the other way of operating with formulas, *subproof generation*. This is to

put the main portion of the proof on "temporary hold" while attempting to show that some other formula is provable (provable based on what has already gone on in the proof so far). When one starts such a subproof one is allowed to make a temporary assumption (the particular assumption being determined by what formula one is trying to prove), and to use this assumption together with earlier parts of the proof to try to prove the subproof's goal formula. When this subproof call succeeds, then the formula which was to be proved becomes a part of the outside, main proof; but the portion of the proof which justified our claim that the subgoal is provable is no longer available to the main proof. (The reason for this is that this portion depends on the assumption that was made, and that assumption is no longer valid.) Since the main conclusion to be proved is itself considered a subproblem, the problem is solved when the main subgoal is proved.

The method discussed later (and the examples presented) concerns restating natural deduction proofs as a natural language argument. Therefore we need to give enough information about the natural deduction system employed so that one can follow the natural language explanations. The underlying logic system used here is that of Kalish, Montague & Mar [1980], which has been implemented as a program called THINKER [Pelletier 1982,1987]. In a natural deduction system there are many rules of inference; the retention of the "natural form" requires that there needs to be rules describing what can be done with each different type of formula. Writing a natural deduction automated theorem proving system is largely a matter of organizing the application of all these rules so as to efficiently generate proofs. The rules of inference of Kalish, Montague & Mar (as modified for THINKER) are the following. Each rule (except for REFL) has some preconditions in terms of formulas that must already be in the proof and must be *antecedent* (a technical term explained below). These preconditions are stated to the left of the '==>'. When these preconditions are met, the formula to the right of the '==>'

may be introduced into the proof (along with an *annotation* -- a justification in terms of Rule of Inference employed and the location [line numbers in the proof] of the preconditions). Some rules take more than one form, as indicated here; the name of the Rule is given in the table, and its abbreviation (which is found in the proofs) is indicated in bold.

RULE	NAME
$\Phi \Longrightarrow \Phi$	**R** (**R**epetition)
$\Phi \Longrightarrow \neg\neg\Phi$	**DN** (**D**ouble **N**egation)
$\neg\neg\Phi \Longrightarrow \Phi$	**DN**
$(\Phi \& \Psi) \Longrightarrow \Phi$	**S** (**S**implification)
$(\Phi \& \Psi) \Longrightarrow \Psi$	**S**
$(\Phi \rightarrow \Psi), \Phi \Longrightarrow \Psi$	**MP** (**M**odus **P**onens)
$(\Phi \rightarrow \Psi), \neg\Psi \Longrightarrow \neg\Phi$	**MT** (**M**odus **T**ollens)
$\Phi, \Psi \Longrightarrow (\Phi \& \Psi)$	**ADJ** (**Adj**unction)
$(\Phi \vee \Psi), \neg\Phi \Longrightarrow \Psi$	**MTP** (**M**odus **T**ollendo **P**onens)
$(\Phi \vee \Psi), \neg\Psi \Longrightarrow \Phi$	**MTP**
$(\Phi \rightarrow \Psi), (\Psi \rightarrow \Phi) \Longrightarrow (\Phi \leftrightarrow \Psi)$	**CB** (**C**onditionals to **B**iconditional)
$(\Phi \leftrightarrow \Psi) \Longrightarrow (\Phi \rightarrow \Psi)$	**BC** (**B**iconditional to **C**onditional)
$(\Phi \leftrightarrow \Psi) \Longrightarrow (\Psi \rightarrow \Phi)$	**BC**
$\Phi \Longrightarrow (\Phi \vee \Psi)$	**ADD** (**Add**ition)
$\Phi \Longrightarrow (\Psi \vee \Phi)$	**ADD**
$\neg(\forall\alpha)\Phi \Longrightarrow (\exists\alpha)\neg\Phi$	**QN** (**Q**uantifier **N**egation)
$\neg(\exists\alpha)\Phi \Longrightarrow (\forall\alpha)\neg\Phi$	**QN**
$(\forall\alpha)\neg\Phi \Longrightarrow \neg(\exists\alpha)\Phi$	**QN**
$(\exists\alpha)\neg\Phi \Longrightarrow \neg(\forall\alpha)\Phi$	**QN**
$(\forall\alpha)\Phi \Longrightarrow \Phi'$	**UI** (**U**niversal **I**nstantiation)
$(\exists\alpha)\Phi \Longrightarrow \Phi'$	**EI** (**E**xistential **I**nstantiation)
$\Phi' \Longrightarrow (\exists\alpha)\Phi$	**EG** (**E**xistential **G**eneralization)
$\Phi\alpha, \alpha=\beta \Longrightarrow \Phi\beta$	**LL** (**L**eibniz's **L**aw)
$\Phi\alpha, \neg\Phi\beta \Longrightarrow \neg\alpha=\beta$	**NEGID** (**NEG**ated **Id**entity)
$\Longrightarrow \alpha=\alpha$	**REFL** (**Refl**exivity of Identity)

In the rules UI, EI, and EG, Φ' is the result of replacing all free occurrences of α in Φ (that is, the ones that are bound by the quantifier phrase in $(\forall\alpha)\Phi$ or $(\exists\alpha)\Phi$) with some term (constant or variable) in such a way that if it is a variable it does not become bound by any other quantifier in Φ'. Furthermore, in the case of EI, the new term must be a variable, and this variable must be entirely new to the proof as thus far constructed. In the

rules LL and NEGID, the relationship between $\Phi\alpha$ and $\Phi\beta$ is that some free occurrences of α in $\Phi\alpha$ can be replaced by β and these occurrences of β will be free in $\Phi\beta$. The fact that the rule REFL has no preconditions means that a self-identity statement can be introduced anywhere in the proof.

An argument in general has premises and a conclusion. In the Kalish, Montague & Mar system, premises can be entered into a proof at any time (with the annotation PREM). What makes natural deduction systems distinctive is the idea of *subproofs* -- "smaller" or "simpler" subproblems which, if proved, can be "put together" to constitute a proof of the main problem. Obviously, a major factor in natural deduction theorem proving (whether automated or not) is the timely selection of appropriate subproofs to be attempted. In the Kalish, Montague & Mar system, one indicates that one is about to attempt a subproof of a formula Φ by writing "show Φ". (This is called "setting a (sub)goal", and the line in the proof which records this is called a show-line). Since the main conclusion to be proved, C, is itself considered a subproblem, the first line of a proof will be "show C". One is allowed to write "show Φ" for any Φ at any further stage of the proof. Intuitively, one is always "working on" the most recently set subgoal -- although one can of course set a further sub-subgoal (and then "work on" that newly set sub-subgoal). The formula following the "show" on one of these lines that indicates a subproof is not really part of the proof proper (until it has been proved). The technical term for this is that it is *not antecedent*, and the effect of being not antecedent is that this formula is unavailable for use in the Rules of Inference.

Setting a subgoal allows one to make an *assumption*. The form of the assumption depends on the form of the formula whose proof is being attempted, and these assumptions can only be made on the line immediately following a show line. (They are annotated ASSUME). There are three types of assumptions allowed:

show $(\Phi \rightarrow \Psi)$
  Φ ASSUME

show $\neg\Phi$
  Φ ASSUME

show Φ
  $\neg\Phi$ ASSUME

The final concept required here is that of *(sub)proof completion* —the conditions under which a (sub)goal can be considered to have been proved. The following is a summary of all the ways to complete a subproof. We suppose that the last portion of the proof so far constructed has the form:

show Φ
  X_1
  ...
  ...
  X_n

Then we can change this part to ("complete the subproof"/"box and cancel"):

*show Φ
| X_1
| ...
| ...
| X_n

if (a) There are no "uncancelled shows" amongst $X_1 \ldots X_n$, and (b) one of the following situations hold: (b_1) Φ occurs "unboxed" amongst $X_1 \ldots X_n$, (b_2) both Θ and $\neg\Theta$ occur "unboxed" amongst $X_1 \ldots X_n$ for some formula Θ, (b_3) Φ has the form $(\Psi_1 \rightarrow \Psi_2)$ and Ψ_2 occurs "unboxed" amongst $X_1 \ldots X_n$, or (b_4) Φ has the form $(\forall\alpha)\Phi\alpha$, $\Phi\alpha$ occurs "unboxed amongst $X_1 \ldots X_n$, and α is not free in any line antecedent to this show line.

When this happens, the lines $X_1 \ldots X_n$ are said to be *boxed* (indicated by the vertical scope line) and are thus no longer antecedent (in the technical sense), while the "show" is said to be *cancelled* (indicated by the *-sign) and the formula Φ is not antecedent (in the technical sense). The boxed lines $X_1 \ldots X_n$ constitute a proof of Φ, but having been used in establishing this goal, they are no longer valid to use (i.e., no longer antecedent in the technical sense) -- the reason being that they may have "depended on" an assumption

made during that subproof and this assumption is no longer in force now that the goal has been proved. Two things should be noted. First, when a subgoal is proved, *all* lines after the show line are boxed -- even if the "reason for cancelling" is not the last line of the proof as thus far constructed. (This is required for soundness). Secondly, although it is common and expected that the method of the proof completion will "match" the type of assumption made -- eg., assuming the antecedent of a conditional should allow one to cancel because one generated the consequent of the conditional -- this is not required. The Kalish, Montague, Mar system is sound even if we allow any "way to start a subproof" to be ended with any "way to complete a subproof".

A proof of Φ from a set of premises Γ is a sequence of lines of formulas and scope lines constructed in accordance with the above, in which Φ occurs unboxed and in which there are no uncancelled show lines. The following are four example proofs to show what proofs look like in the Kalish, Montague, Mar system:

1	*show $(\forall x)(Fx \rightarrow Gx) \rightarrow ((\forall y)Fy \rightarrow (\forall z)Gz)$	
2	| $(\forall x)(Fx \rightarrow Gx)$	ASSUME
3	| *show $(\forall y)Fy \rightarrow (\forall z)Gz$	
4	| | $(\forall y)Fy$	ASSUME
5	| | *show $(\forall z)Gz$	
6	| | | $Fz \rightarrow Gz$	2,UI
7	| | | Fz	4,UI
8	| | | Gz	6,7MP

1	*show $(P \vee \neg\neg\neg P)$	
2	| $\neg(P \vee \neg\neg\neg P)$	ASSUME
3	| *show $\neg P$	
4	| | P	ASSUME
5	| | $(P \vee \neg\neg\neg P)$	4,ADD
6	| | $\neg(P \vee \neg\neg\neg P)$	2,R
7	| $\neg\neg\neg P$	3,DN
8	| $(P \vee \neg\neg\neg P)$	7,ADD

```
1   *show (P→Q)→(¬Q→¬P)
2   | (P→Q)     ASSUME
3   | *show (¬Q→¬P)
4   | | ¬Q       ASSUME
5   | | ¬P       2,4MT
```

```
1   *show (Faaa & a=b)→Fbab
2   | Faaa & a=b     ASSUME
3   | Faaa           2,S
4   | a=b            2,S
5   | Fbab           2,3LL
```

The current version of THINKER makes some minor changes to the preceding Kalish, Montague, Mar explanation. These alterations are mentioned so as to be able to display the output of THINKER below. The main change is in the representation of subproof completion, where now there is an annotation-justification printed on the "show" line. For example, where we might before have seen

```
n.     *show P→Q
n+1.   |  P  ASSUME
       |  ...
       |  ...
m      |  Q
```

```
n.  *show P
    |  ...
m.  |  Q
    |  ...
p.  |  ¬Q
```

```
n.  *show (∀x)Fx
    |  ...
    |  ...
    |  ...
m.  |  Fx
```

we now will see, respectively

```
n.     *show P→Q  n+1,mImpInt
n+1.   |  P  ASSUME
       |  ...
       |  ...
m      |  Q
```

```
n.  *show P  m,pContra
    |  ...
m.  |  Q
    |  ...
p.  |  ¬Q
```

```
n.  *show (∀x)Fx  m,UG
    |  ...
    |  ...
    |  ...
m.  |  Fx
```

The annotation "(n+1),m ImpInt" indicates that line n was cancelled by "implication introduction" using lines (n+1) and m. The annotations "Contra" and "UG" indicate that the show line was cancelled by having found a contradiction on the stated lines or by universal generalization (which presumes that x was not free in any line antecedent to line n). Another feature of this style is that the "reason for being able to cancel" is no longer required to be within the subproof developed beneath the show line, so long as it *could have been*. For example, the following is allowed

k. Q
...
n. *show P k,m Contra
| ...
| ...
m. | ¬Q

(so long as line k is antecedent). In Kalish, Montague, Mar one can cancel a universally quantified show line, "show $(\forall\alpha)\Phi\alpha$", if one can derive *exactly* the formula on the show line, but without the quantifier phrase; that is, if one can derive $\Phi\alpha$. There is a restriction that the variable of quantification, α, must not be free in any lines antecedent to the show line. This restriction is to ensure that the variable is "really arbitrary" and therefore that having generated $\Phi\alpha$ really means that "Φ is true of an arbitrary thing" so that we are entitled to consider $(\forall\alpha)\Phi\alpha$ proved. In THINKER, a special list of "arbitrary variables" is kept, and a universally quantified goal is proved if the unquantified formula can be generated with one of these special variables replacing occurrences of variables bound by the quantifier. Thus

show $(\forall x)(Fx \rightarrow (Gy \& Hx))$

can be cancelled if

$Fr_0 \rightarrow (Gy \& Hr_0)$

is generated, where r_0 is not free in antecedent lines of the proof before the setting of the this goal.

The proofs given earlier in this section would be done this way in THINKER:

```
1  *show (∀x)(Fx →Gx)→((∀y)Fy →(∀z)Gz)        2,3 ImpInt
2  |  (∀x)(Fx →Gx)                            ASSUME
3  |  *show (∀y)Fy →(∀z)Gz                    4,5 ImpInt
4  |  |  (∀y)Fy                               ASSUME
5  |  |  *show (∀z)Gz                        6 UG
6  |  |  |  *show Gr0                         7,8 MP
7  |  |  |  |  Fr0→Gr0                        2 UI
8  |  |  |  |  Fr0                            4 UI
```

```
1  *show (P ∨ ¬¬¬P)          6 ADD
2  |  ¬(P ∨ ¬¬¬P)            ASSUME
3  |  *show ¬P               2,5 Contra
4  |  |  P                   ASSUME
5  |  |  (P ∨ ¬¬¬P)          4 ADD
6  |  ¬¬¬P                   3 DN
```

```
1  *show (P→Q)→(¬Q→¬P)        2,3 ImpInt
2  |  (P→Q)                  ASSUME
3  |  *show (¬Q→¬P)          4,5 ImpInt
4  |  |  ¬Q                  ASSUME
5  |  |  ¬P                  2,4 MT
```

```
1  *show (Faaa & a=b)→Fbab    2,5 ImpInt
2  |  Faaa & a=b              ASSUME
3  |  Faaa                    2 S
4  |  a=b                     2 S
5  |  Fbab                    2,3 LL
```

Natural deduction is used for automated theorem provers because it is closely related to the way that humans would do theorem proving themselves. The reason behind this is that natural deduction uses many different inference rules as would a human and thus is easier to explain and understand. On the other hand, a resolution based theorem prover uses only one inference rule to produce its proofs. The one inference rule approach is difficult to explain since it does not correspond to the way a human

would attempt to prove a statement. Therefore, we will deal only with natural deduction proofs.

1.4 Multiple Explanations for Proofs

Natural deduction proofs generated by an automated theorem prover can differ significantly. For example, a proof could be a straight line type of proof where each step follows the proceeding step and few subproofs are required. On the other hand, the proof could be a nested type of proof in which there are several subproofs that need to be proved. There are many other ways in which they can vary. When explaining the proofs generated, the fact that the proofs are not all the same should be taken into account and the ability to generate different explanations should be an available option.

When a human is producing a natural deduction proof there are three methods that they can employ. The first would be to start with the known true formulas and then use an inference rule to create more of these true formulas. This method would continue in a effort to produce the main goal. This is called forward reasoning. Secondly, the human may begin with the main goal and split it into several subgoals to prove while making relevant assumptions. The splitting of the subgoals continues until a subgoal can no longer be split or it is found to be a premise or follows from the premises and assumptions. The main goal is proved when all its subgoals are proved. This reasoning is called backward. A combination of these two types is used sometimes proving subgoals top-down and others bottom-up and then combining their proofs into one. Since a human produces proofs in these ways an automated reasoner should be able to explain its proofs as such to give the user the explanation in fashion of proving theorems that they are familiar with.

If applications were developed for a single user the explanation facility could be tailor fit to that specific user. This is generally not the case. Applications are produced to be used by many different people. The users may have many levels of understanding when it comes to the proofs generated by the system. This could range from those who are logicians and do not need any natural language explanation facility to naive users who would like the proof completely translated into English. The explanation facility should be able to accommodate the needs of many users and thus be able to produce many different explanations for a single generated proof.

Another way that multiple explanations would be beneficial is that one user may desire several different explanations of a single proof to aid in his/her understanding. One explanation may not be enough to alleviate all the misinterpretations that a user might have. If a section of the proof is hard for the user to understand it may be of some help if they could look at a few different explanations of the same part of the proof. Where as if only one explanation is produced and it does not help the user understand the proof no further help can be given.

2. Background of Explaining Natural Deduction Proofs

2.1 Introduction

There are several papers that have dealt with generating natural language explanations from natural deduction proofs. This chapter describes four of the papers in the field.

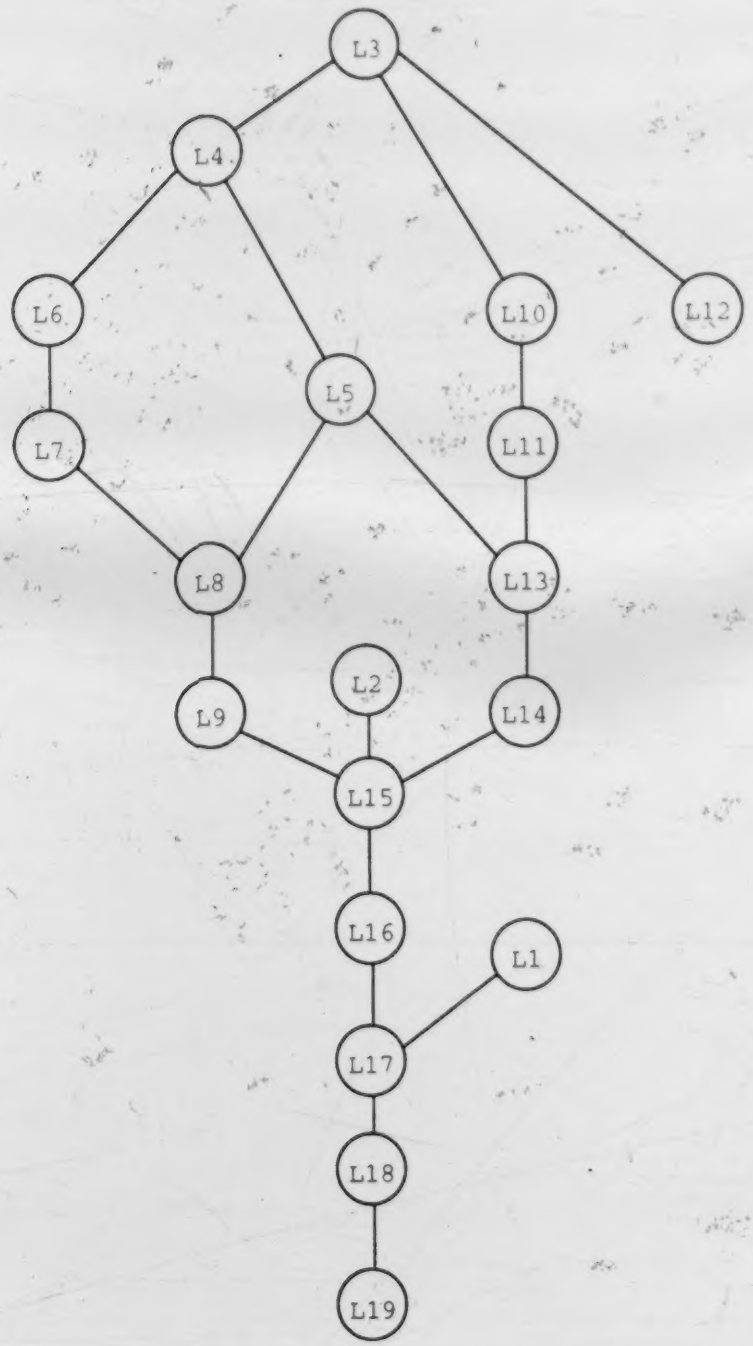
2.2 Chester's Paper

[Chester76] was one of the first papers to discuss the explanation of natural deduction proofs in natural language. The paper discusses the program EXPOUND that produces a translation of a formal proof into English. There are four stages to the generation of the English explanation. The first step is to create a graph that represents the inferential relationships between the lines in the proof. In the next two stages this graph is used to make an outline of the final text. The second step is to group the lines into paragraphs. The paragraphs are then put into a linear order and introductory paragraphs are inserted to explain how the paragraphs are related. Finally, EXPOUND generates the English text by explaining how each line is obtained from the previous lines in the output. The output of the program is the statement of the theorem and the translation of the proof into English. An example proof from [Chester76] is given below. The inference rules used by EXPOUND are the same as those for THINKER with a few exceptions: PR is similar to ASSUME and PREM, CD corresponds to IMP INT, and the rule CQ is called QN by THINKER. The one rule that is not present in THINKER is TF which is truth-functional inference, in other words, infer Y from $X_1 X_2 ....$ because $(X_1 \& X_2 \& ...) \rightarrow Y$ is a tautology.

L1	$\forall x((Fx \& Gx) \rightarrow \exists x(Fx \& \neg Gx)$	(PR)
L2	$\forall x(Fx \rightarrow Gx) \vee \forall x(Fx \rightarrow Hx)$	(PR)
L3	$\forall x((Fx \& Hx) \rightarrow Gx)$	(PR)
L4	$\exists x(Fx \& \neg Gx)$	(PR)
L5	$Fc \& \neg Gc)$	(EI L4)
L6	$\forall x(Fx \rightarrow Gx)$	(PR)
L7	$Fc \rightarrow Gc$	(UI L6)
L8	$Gc \& \neg Gc$	(TF L5 L7)
L9	$\forall x(Fx \rightarrow Gx) \rightarrow (Gc \& \neg Gc)$	(CD L6 L8)
L10	$\forall x(Fx \rightarrow Hx)$	(PR)
L11	$Fc \rightarrow Hc$	(UI L10)
L12	$(Fc \& Hc) \rightarrow Gc$	(UI L3)
L13	$Gc \& \neg Gc$	(TF L5 L11 L12)
L14	$\forall x(Fx \rightarrow Hx) \rightarrow (Gc \& \neg Gc)$	(CD L10 L13)
L15	$Gc \& \neg Gc$	(TF L2 L9 L14)
L16	$\exists x(Fx \& \neg Gx) \rightarrow (Gc \& \neg Gc)$	(CD L4 L15)
L17	$\neg \forall x((Fx \& Gx) \rightarrow Hx)$	(TF L1 L16)
L18	$\exists x(Fx \& Gx \& \neg Hx)$	(CQ L17)
L19	$\forall x((Fx \& Hx) \rightarrow Gx) \rightarrow \exists x(Fx \& Gx \& \neg Hx)$	(CD L3 L18)

The first step taken by Chester's program EXPOUND to generate the translations of the formal proofs into English is the creation of a graph representing how the lines of the proof relate to each other. This graph is created from a partial order of the lines in the proof. This partial order is defined by four conditions. These conditions describe the basic facts about how the lines in the proof are usually presented. The first condition is that the reasons for a line are given before the conclusion is drawn from them. The second asserts that when a proof is started from some assumption x and a subproof can not be completed without using a successor to x (line y such that x>y, where '>' is the partial order on lines in the proof and x>y means line x comes before line y) then the entire subproof is nested in in the original proof. The third condition used to generate the graph states that when a constant is introduced by existential instantiation then this line

Figure 1. First graph of the example proof.

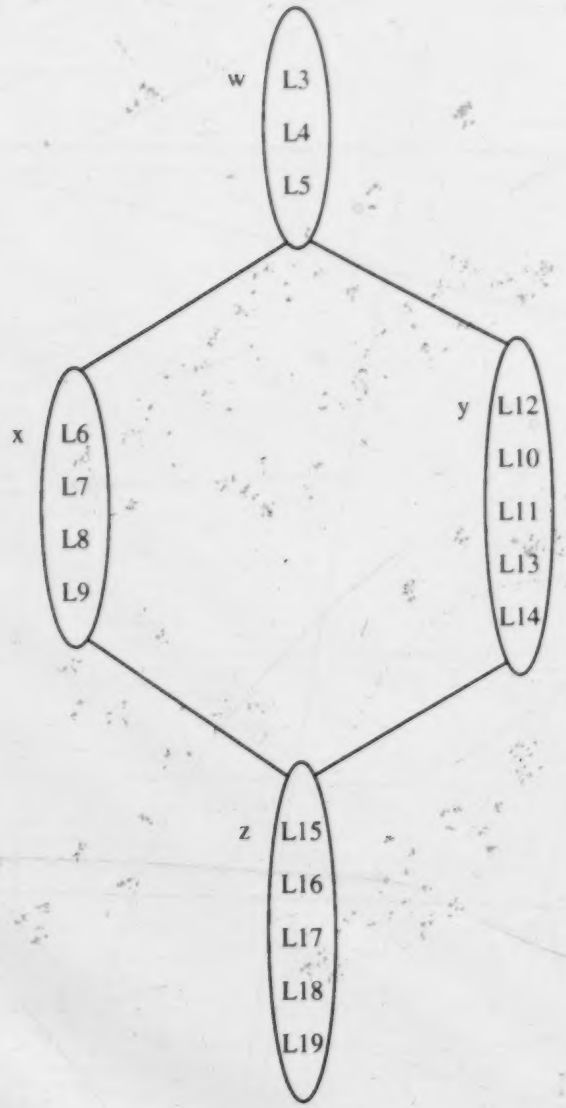


must proceed all other lines that reference it. The final condition is simply that the partial ordering has the transitive property. (x>y and y>z we can say x>z) The partial ordering is represented internally by a graph and this is what is used in the next two steps. The graph used to represent the example is seen in figure 1.

The second step in the generation process is to combine the nodes of the graph into sequences of lines that outline the paragraphs that will appear in the output. After this process the paragraphs may not be in a linear order so EXPOUND will list the nodes in a compatible order with the partial order. At this point the program may insert some introductory paragraph nodes to the graph. These are used to clarify the relationships between the other nodes. The total proof and each subproof are examined individually to determine where these types of nodes are needed. The introductory paragraphs include a conclusion and a list of lemmas that must be shown to prove the conclusion. The final graph showing the lines of the example grouped into paragraphs is in figure 2. In this case EXPOUND inserts an introductory paragraph node consisting of conclusion L15 and lemmas L9 and L14 right after node w, making the final order w, introductory paragraph, x, y, and z.

Once the outline of the final text to be produced is done the translation into English sentences is attempted. The lines in a paragraph are generated to explain how the lines in the proof are related. Each line of the proof that is to be in the paragraph is examined and a sentence is generated depending on the inference rule and the previous sentences in the discourse. Not every line in the paragraph may have a corresponding sentence, some lines that are easily deduced by the reader may be left out. The sentences themselves are generated by using some specific lexical information which is passed to EXPOUND. This information is about the predicates that appear in the proof. The lexical information includes the number of arguments, prepositions for the arguments, and several phrases

Figure 2. Final graph showing lines grouped into paragraphs.



that the predicate can be translated to. Using the information, phrases for atomic formulas, predicates and their arguments, and quantified formulas are generated. These types of phrases are combined using connective phrases that correspond to the connectives found within the formulas of the proof. The lexical information given to EXPOUND was as follows.

Predicate	Property	Value
F	arguments	x
	+active form	is a worker
	-active form	is not a worker
	gender	M
	syntactic type	noun
G	arguments	x
	+active form	signed the contract
	-active form	did not sign the contract
	syntactic type	clause
H	arguments	x
	+active form	is in the union
	-active form	is not in the union
	syntactic type	phrase
Universe	arguments	x
	+active form	is a person
	gender	M
	syntactic type	noun

The final output from the program EXPOUND for the example includes the statement of the theorem and the explanation of the proof. The output is shown below.

THEOREM. *Suppose that if every worker who signed the contract is in the union then some worker did not sign the contract. Suppose moreover that either every worker signed the contract, or every worker is in the union. Then if every worker in the union signed the contract then some worker who signed the contract is not in the union.*

PROOF. *Suppose that every worker in the union signed the contract. Suppose moreover that some worker did not sign the contract. Let w denote such a worker who did not sign the contract.*

We want to show that a contradiction follows. This we shall do by considering the following 2 cases.

Suppose that every worker signed the contract. Now a contradiction follows, since by assumption w is a worker and he did not sign the contract, and if he is a worker then he signed the contract. Thus if every worker signed the contract then a contradiction follows.

Suppose that every worker is in the union. But a contradiction follows, as by assumption w is a worker and he did not sign the contract, and if he is a worker then he is in the union, and if he is a worker and he is in the union then he signed the contract. Thus if every worker is in the union then a contradiction follows.

Because by hypothesis either every worker signed the contract, or every worker is in the union, and we have shown that if every worker signed the contract then a contradiction follows, and if every worker is in the union then a contradiction follows, a contradiction follows. Thus if some worker did not sign the contract then a contradiction follows. This and the fact that by hypothesis if every worker who signed the contract is in the union then some worker did not sign the contract imply that not every worker who signed the contract is in the union. In other words some worker who signed the contract is not

in the union. Therefore if every worker is in the union signed the contract then some worker who signed the contract is not in the union.

Most of the effort in [Chester76] was in the ordering of the lines for presentation in an English language fashion. In general the explanation of the proof is presented as forward chaining. In other words the premises and assumptions are handled first and then the lines that can be derived from them are explained, leading to the final goal. The introductory paragraphs give a hint of a top-down type explanation by describing what is to be explained and how it is to be done before the explanation of the subgoals takes place. This effort in explanation of natural deduction proofs creates only one explanation type for any one proof. The only variation for a single proof would be in the random selection of phrases with which to explain certain inference rules. All Chester's explanations are the same with a backward reasoning explanation for the introductory paragraphs and a forward explanation of the main body of the proof.

2.3 McDonald's Paper

Another method for generating natural language explanations of natural deduction theorem proving is in [McDonald85]. This work is about generating natural language from several different message formats, natural deduction proofs being one of these formats. Most of the work done is to the piece of the system that is called the linguistic component. This component contains two parts which McDonald calls transducers. The first transducer takes the message passed to it for translation into English and produces a surface structure level linguistic representation of the utterance to be produced. This structure is the "working" data structure that will be used in creating the output. The second transducer takes this structure and translates it into the English text.

The generation of language is thought of as a decision making process. The set of possible consistent decisions are determined by the language's grammar. Therefore, the linguistic component is where the decisions take place and are acted on. The first transducer is the decision maker. Decisions are made on how to realize the individual elements of the message, through the selection of particular surface structure phrases. The second transducer has the job of executing the decisions that were made in the first. They are executed by interpreting the surface structure that the first transducer developed as a program of linguistic actions that are to be performed to generate the desired discourse.

To make the system work for a specific type of message, such as natural deduction proofs, two things must be customized. The first is the dictionary. The dictionary holds the information about how to interpret the message element by element, to determine its linguistic correspondences and relevant substructure. The second part which must be made specific to the application are the interface functions. These functions have two purposes. First, to link elements of a message to the dictionary entries. And secondly, to answer some linguistic questions such as "person and number" for the message elements.

Natural deduction proofs are used as one type of message that can be fed into this system. With this, lines of the proof are passed in sequence to the program. The English text that was selected for earlier lines provides a discourse context to narrow the choices that can be made for the current line. Things such as subsequent references to constants, variables used as generic references, and predicates and formulas used as description are dependent on previous lines for their realization. A large part of the English text is devoted to the explanation, guided by the inference rules, of how each line is related to the earlier ones. The program processes a proof by realizing its formulas and subformulas as the lines are read in. This realization is not mechanical; a context sensitive decision is made as to how (or whether) the major connective or inference rule is to be

translated. It also decides which (if any) of the subelements of the formula are to be used in this realization. Certain lines of the proof may be omitted if it is felt that the step could be made automatically by the audience.

This system has many pleasant features in the explanation of natural deduction proofs. Quantifiers can be translated into determiners of their variables to give phrases like "some barber" or "everyone" rather than producing phrases such as "for every" or "there exists a person such that". The system can use functional labels such as "an assumption" or "a contradiction" and use these phrases as references to complete formulas. For some connectives the realization will differ depending on the context. For example, the connective $\leftrightarrow$ can be realized simply as "if and only if" in a formal context but when seen as a restriction on a variable it can be expressed as a relative clause "everyone who" or "all and only those men who $\cdots$". The program uses phrases like "leads to" for linking lines to one another when the current line was deduced from the previous. The system also has a special monitoring routine it invokes to help identify and avoid possible ambiguities that may be introduced into the discourse. A way to avoid the ambiguities is with the use of parentheses to forgo any misinterpretation of the intended scope. An example proof and the output produced is given below.

line1: premise

$\exists x$ (barber(x) & $\forall y$(shaves(x,y) $\leftrightarrow \neg$shaves(y,y)))

line2: existential instantiation (1)

barber(g) & $\forall y$(shaves(g,y) $\leftrightarrow \neg$shaves(y,y))

line3: tautology (2)

$\forall y$ shaves(g,y) $\leftrightarrow \neg$shaves(y,y)

line4: universal instantiation (3)

shaves(g,g) $\leftrightarrow \neg$shaves(g,g)

line5: tautology (4)

shaves(g,g) & ¬shaves(g,g)

line6: conditionalization (5,1)

∃x (barber(x) & ∀y(shaves(x,y) ↔ ¬shaves(y,y))) ⊃ (shaves(g,g) & ¬shaves(g,g))

line7: reductio-ad-absurdium (6)

¬∃x (barber(x) & ∀y(shaves(x,y) ↔ ¬shaves(y,y)))

Assume that there is some barber who shaves everyone who doesn't shave himself (and no one else). Call him Giuseppe. Now, anyone who doesn't shave himself would be shaved by Giuseppe. This would include Giuseppe himself. That is, he would shave himself, if and only if he did not shave himself, which is a contradiction. This means that the assumption leads to a contradiction. Therefore, it is false, there is no such barber.

This system generates some easily understandable explanations of some natural deduction proofs. It goes to great length to be able to generate extremely natural sounding explanations. The program produces the lines of the explanation in a generally forward fashion. The simple lines such as assumptions and premises are explained first and then the explanation build on these towards explaining the final goal last. The system produces only one explanation for each proof that it is given. No variation can be made to the order in which the lines of the proof are presented.

2.4 Felty and Hager's Paper

Another paper that dealt with generating explanations of natural deduction proofs is [Felty88]. This paper deals with taking a natural deduction theorem prover which is expanded to handle proofs in modal logic and generating explanations of these types of proofs. The explanation facility is simple yet flexible to be able to generate different kinds of explanations to meet the needs of different users. The explanations can take on

numerous forms depending on taste, background, and the type of information to be conveyed.

The system uses a Gentzen proof style sequent system. A sample proof of a statement about a reflexive relation R and a predicate P is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{R(a,a)\rightarrow R(a,a) \qquad P(a)\rightarrow P(a)}{R(a,a),R(a,a)\supset P(a)\rightarrow P(a)}\,forwardchain}{R(a,a),\forall y(R(a,y)\supset P(y))\rightarrow P(a)}\,all\text{-}L}{\forall w\ R(w,w),\forall y(R(a,y)\supset P(y))\rightarrow P(a)}\,all\text{-}L}{\forall w\ R(w,w)\rightarrow \forall y(R(a,y)\supset P(y))\supset P(a)}\,imp\text{-}R}{\forall w\ R(w,w)\rightarrow \forall x[\forall y(R(x,y)\supset P(y))\supset P(x)]}\,all\text{-}R}{\rightarrow \forall w\ R(w,w)\supset \forall x[\forall y(R(x,y)\supset P(y))\supset P(x)]}\,imp\text{-}R$$

The proofs that are used in this system are represented by a recursive term structure. The proof term for the example above is:

imp_r(all_r(imp_r(all_l(all_l(forwardchain(axiom(R(a,a)),axiom(P(a))))))))

The explanations are generated by mapping from these terms to text strings. The form and content of the explanation depend on the information extracted from the corresponding proof term and the way in which that information is mapped to strings of text. Different explanations are generated by different kinds of mappings. Each inference rule has a corresponding function that takes the explanation of the premises and puts them together, with the addition of more text, to construct the explanation. To explain a line in the proof the function takes the text that explains the lines that it depends on and adds some more text to generate an explanation for the current line. In these explanations only the inference rules are explained the formulas are not lexicalized. In other words, the formulas are left as symbols and are not expanded into natural language. The explanation

produced for the example above is:

Assume $\forall w\ R(w,w)$. *Assume* $\forall y(R(a,y) \supset P(y))$. *Let* $w = a$ *in* $\forall w\ R(w,w)$. *Let* $y = a$ *in* $\forall y(R(a,y) \supset P(y))$. *By modus ponens we have* $P(a)$. *Since* a *was arbitrary we have* $\forall x[\forall y(R(x,y) \supset P(y)) \supset P(x)]$.

This explanation facility is produced with a particular domain, modal logic, in mind. If proofs are specific to a particular domain, the explanations should be presented in terms of concepts that have meaning within that domain. For example, when explaining proofs about inheritance the explanation should use terms that relate to inheritance rather than the basic logic operations. The explanations of the modal logic proofs are to be in terms of the modal logic domain. To add modal explanations to the system, text generation functions will be added whose contribution to the explanations depend on the meaning of the corresponding modal operators. This contribution will also vary as to how much detail is desired and the context.

The explanations of the modal logic proofs can be produced at several levels of detail. One level of explanation for modal proofs would be for those users who are familiar with the rules of modal logic. The details about how possible worlds are related would be left out of the explanation since this would add unnecessary clutter. The reader is already familiar with these concepts. This would lead to a skeletal explanation of the modal logic proof. A more detailed version of explanation would have to be given for those who do not understand the inference rules and axioms of modal logic. To explain the first-order translation of the proof would be too general. An algorithm should be developed that explains the proof at a "deeper" level, specialized to possible world semantics. With this type of explanation the text generating functions for all the inference rules must be changed slightly to accommodate information about situations. These expanded explanations give two additional types of information. First, there is more

detail about the chain of inference. And secondly, the information of the possible world ornamentations is used to give an explicit reference to how the situations are related.

This work recognizes that it may be necessary to generate more than one type of explanation for a single proof. These different levels are generated by changing the mappings from proof structures to text strings. This may require the changing of the functions that do the explanation. The explanations themselves are mainly goal-directed. This means that the main goal is stated and the lines it is dependent on are explained afterwards. The inference rules are the only thing that are lexicalized in these explanations. The formulas within the lines of the proof are left as they were generating an explanation that is simple to follow if the reader is able to understand the formulas. On the other hand if the user does not even understand the formulas then this type of explanation would not be of much use.

2.5 Huang's Work

Some more work that deals with the generation of natural language explanations of natural deduction proofs is [Huang89a, Huang89b, Huang90]. These works deals with the transformation of the "logical level" proof to the "conceptual level" proof, that is easier to explain. The inference rules in the logical level of proof are in general very small. The goal is to use inference steps the size that human mathematicians usually prefer. For example, suppose we have the concept of "subset" encoded as follows:

$$U \subseteq F \leftrightarrow \forall x \; x \in U \rightarrow x \in F$$

Now everyone who has a standard mathematical training will find it natural to deduce $a \in F$ given $U \subseteq F$ and $a \in U$ as facts. Even if they are not familiar with set theory they will reason at this level of abstraction. To deduce the same inference in a Gentzen-style proof would require eight lines. The point is that the conciseness found at the conceptual level

is not really based on the person's competence in the particular area, and the tediousness at the logical level is not caused by the machine's inability to find elegant proofs. Rather, the reasoning is simply taking place at different levels of abstraction.

If one concentrates on a special area of mathematics, compound rules similar to the rule above can be distinguished. Indeed, they usually correspond to one application of either an axiom or a theorem. A simple but powerful method was developed to derive domain-specific compound inference rules from axioms and theorems. These new rules are then incrementally added to the calculus as new theorems or intermediate results or lemmas are proved.

The next problem, after the derivation of the domain-specific compound rules, is how to shorten the original Gentzen proof. This is done in a brute force method where, in a bottom-up direction, each proof line is checked to see if there is an applicable compound rule. If a rule is applicable, the inference rule is changed on that line and the pointers to the reason lines are changed as needed. After these tests are complete the proof is searched again and any line not used in the new proof is deleted.

Once the Gentzen proof that was input has been raised to a conceptual level proof, the system proceeds to translate them into so-called message sequences. To do this the proof must first be organized into an ordered proof tree. The tree has the conclusion of the proof at the root and the leaf nodes contain assumptions and premises. The tree is ordered first using the parent-children relation that is based on the inference rules. Ordering of the children is dependant on both structural and pragmatic constraints. The simple logical constraint used for ordering is that the reasons must appear before conclusions. This is easily satisfied by using a post-order traversal of the proof tree to produce a total order of the message segments.

The second type of constraint that must be satisfied for the ordering of the message segments is a structural constraint. Some inference rules that require more than one reason impose an inherited order on their reasons. In other words the reasons are almost always derived in a particular order, although there is nothing logical forbidding doing it another way round. This order is closely related to the forward chaining proof development style that is employed in this system.

The final constraint used for the order of the proof tree is a *focus mechanism*. There are two types of foci that can be used, global and immediate. Global focus refers to the fact that we usually center our attention on a particular object throughout a set of consecutive utterances. This is especially relevant in a proof where properties about an object are usually grouped together in a set of consecutive proof lines before the proof turns to properties of another object. Immediate focus is the way attention shifts or remains the same over two consecutive sentences. The focus mechanism will be tried when more than one choice concerning ordering remains after the application of all other text structuring rules.

These three constraints are integrated to provide a total order. First, an initial proof tree is built from the proof already raised to the conceptual level. The logical constraint is guaranteed to be satisfied and therefore starting from the root node and proceeding in a pre-order manner an attempt is made to enforce an order on the children of every node. At each node the structural constraint rules will be tried. Beside this enforcement of order, some particular nodes of some subproofs are "marked", indicating that these proof lines should appear at the beginning of the corresponding subproof. Because the system works downward from the root this marking frequently provides an initial focus within which to continue. These markings will be used at nodes where no existing structural rules are applicable. For example, suppose there is a node N with children $N_1, N_2, \ldots, N_k$

and within a subtree of N a node n was marked by the previous process. This node n establishes the "initial" focus and the system would like to proceed in this focus space before turning to another. Therefore the child node picked to appear first will be the node with the smallest focus space which includes everything in the focus space of node n. If none of the children can satisfy this condition another node is found in their subtrees and the child tree that contains this node will be chosen to come first in the ordering. Once all the constraints are satisfied the proof tree will be totally ordered.

A simple post-order traversal of the proof tree is used to produce a linearized message sequence that is passed to the tactical component and realized in natural language. During the traversal, decisions about "what to say" must be made at each node, or in other words, for each proof step. The message unit for each proof step is of the format:

<<inference-rule, reasons, proof-line>>

While the proof line will usually be handed over unchanged there are alternative *reference choices* both for the inference rule and the reasons. The three reference choices for the inference rules are:

1. The *explicit* form: this is the case where the inference rule used is indicated explicitly, such as "by definition of unit element" for domain-specific rules and similar translations for Gentzen structural rules.

2. The *omit* form: words such as "thus", or "therefore" will be used.

3. The *implicit* form: the middle ground between explicit and omit. Nothing is said directly as to the inference rule, an implicit hint to the inference rule is given in the translation of the reasons (or the proof line itself). Another way is to use a hint word such as "similarly" if the same rule had been used in the previous line.

Similarly, three reference choices are found for reasons:

1. The *explicit* form: for example, reasons proved "far before" will be repeated explicitly.

2. The *omit* form: reasons can be omitted it they have just been proved or mentioned in other proof steps.

3. The *implicit* form: this is similar to the implicit form used for inference rules, the reasons can be "hinted" implicitly. The translation of the inference rule can be used to hint at the reasons used.

What finally appears at the position "inference rule" in the message format will be one of the following:

1. A name of one of the structural Gentzen rules.

2. A name of a definition or theorem of the current theory level.

3. "omit", for all other cases.

To discover which reference choice will be made for the reasons, structures called proof units must be used. In general every proof structure is a recursive structure of basic components that have been called proof units. Proof units are subtrees that a human mathematician will accept as a subproof. The proof units are assigned a specific type as defined below:

1. The *active proof unit* is the smallest proof unit containing the current node. There is exactly one active proof unit at a time.

1. The *controlling proof unit* is the smallest proof unit containing the active unit. There is exactly one controlling proof unit at a time, except when the active proof unit is an outmost proof unit.

3. *Precontrol proof units* are proof units containing the controlling proof unit.

4. *Closed proof units* are proof units lying before the active proof unit in the ordered proof tree, i.e. nodes in any closed proof unit are those that have been processed when the traverse procedure has reached the current node.

Using these definitions of proof unit types, the reasons can be assigned a contextual status that in turn is used to determine the reference choice for the reasons. The assignment of the contextual status is done as follows:

1. Reasons in the *active* proof unit are structurally close.

2. Reasons in the *controlling* proof unit but not inside any closed units are structurally close.

3. Reasons that are root nodes of immediate subordinate closed proof units are structurally close. Other reasons in a closed proof units are structurally far.

4. Reasons in a precontrol proof unit are far.

The structural distance and the physical distance, defined as the textual distance between the last mention of the reason and the current sentence where the reason is used, are used to determine the reference choice for the reasons as explained below:

1. If a reason is structurally close and near in distance, it will be omited.

2. If a reason is structurally close but far in distance, first try to find an implicit form, if this is not possible, use an explicit form.

3. If a reason is structurally far but near in distance, first try to find an implicit form. If this is not possible, omit it.

4. An explicit form will be used if a reason is both structurally far and far in distance.

The reference choice rules for reason and the reference choice rules for the inference rules are combined to construct an algorithm to produce a message sequence from the ordered tree. The tree is traversed in a pre-order manner applying the two sets of rules at

each non-duplication node. Once all nodes have a completed message unit the system will traverse the ordered proof tree in a post-order fashion to produce the message sequence.

These papers [Huang89a, Huang89b, Huang90] do not describe the actual translation from the message sequence into a natural language explanation. Only one explanation can be produced for any given proof. The explanation itself will be generally a forward chaining method since the message sequence was produced by a post-order traversal of the ordered proof tree.

3. How to Generate Many Explanations

3.1 Explaining Proofs: Dynamics, Formulas, Levels, and Methods

There are four fundamental dimensions along which natural language explanations of natural deduction proofs might vary. These dimensions seem to be independent, and the different varieties of one dimension can pretty freely co-occur with any variety from any of the other dimensions.

The first dimension is a measure of how much one is to explain the strategy behind the construction of the proof vs. a simple recounting of the proof as it stands. That is, this dimension has a dynamic account of the proof construction at one end and a static account of the generated proof at the other. This dimension is particularly salient in natural deduction systems, especially when it comes to the setting of subgoals. At the dynamic side of the explanation we would expect the natural language explanation to say such things as "At this stage of the proof we set the subgoal on line 6 because we notice that line 4 has a certain property and if we could prove line 6 then we could do something to lines 4 and 6. Furthermore it looks like we can prove line 6 because of blah-blah..." At the static side of this dimension we would see explanations such as "At line 6 a particular subgoal was set and it was proved by blah-blah. Once line 6 was proved it was used with line 4 to do something."

The thrust of the program EXPLAIN is to the static side of this dimension. The task that was set was to examine the proof generated by THINKER, and to give an explanation of it. The natural language generator had no private information about THINKER's internal proof development strategies, and therefore was not in a position to try to give

the dynamic type of explanation. To add this ability, the program would have to be able to read the heuristic THINKER used at a particular point from the final proof; and then it would need to have some insight into THINKER author's mind to see why a certain heuristic was thought to be useful at the particular point in the proof that it was employed.

The second dimension along which explanations can vary concerns the amount of explanation that is done for an individual line of the proof. There are three different ways that the formulas can be presented here. First, they can appear as they did in the proof with no change, that is, as uninterpreted formulas. Secondly, since the connectives in the formulas represent well-known English phrases such as: 'if--then', 'and', 'or', 'for every'. We can use this fact to state the various formulas as asserting these kinds of relationships amongst (uninterpreted) simple predicates. Finally, since many applications attribute a particular meaning to the abstract predicate symbols used in the proof, we could use this information to expand the predicatess into English phrases so as to yield a full English language explanation of the proof.

At one end of this dimension the formulas used in the explanation are left in the same form as in the proof. For each line the inference rule would be explained using the formulas needed. This would result in the explanation of a single line such as: "We have shown the biconditional, $(P(r_1,r_1) \leftrightarrow \neg P(r_1,r_1))$, and one side $(P(r_1,r_1))$ is true so we can get the other side $(\neg P(r_1,r_1))$." The explanation explains how the formulas led us to conclude the current line. The formulas themselves are not expanded at all and this explanation would be of little help to someone who did not comprehend them in this format, although it may help some logicians.

The connectives in the formulas represent concepts that are easily translated into English phrases. The midpoint of this current dimension is to have the connectives

translated into English phrases and used to assert relationships amongst (untranslated) predicate symbols. Therefore at this level the inference rules are explained by using phrases that represent the formulas that they act on. An example of this type of explanation of a line is as follows: "Since the biconditional, ($P(r_1,r_1)$ follows if and only if ($P(r_1,r_1)$ is not true)), was shown, and one side is true ($P(r_1,r_1)$), we can conclude the other side (($P(r_1,r_1)$ is false))." This is basically the same explanation for the inference rule, but the formulas on which that rule is applied are now represented by predicate symbols and connecting phrases.

A further step along this dimension towards a complete English explanation of lines in the proof must involve the lexicalization of the predicate symbols. In most applications the abstract predicate and constant symbols are intended to represent some real world relation or item. These mappings can be used to further expand the explanation. The inference rule, the connectives, and also the predicates within the formulas will all be expanded, leading to full English sentences. Using the previous example we can let the predicate symbol "P" represent the concept "x shaves y" and the constant "r_1" stand for a person named Fred. This would lead us to the explanation of the line to be: "Since the biconditional, Fred shaves himself if and only if Fred doesn't shave himself, was shown, and one side is true (Fred is shaved by himself), we can conclude the other side (Fred doesn't shave himself)." This sort of explanation is at one end of the dimension where the complete explanation is in English and there are no further things in the proof line which can be explained.

A proof can be thought of as a tree structure with the nodes of the tree representing lines of the proof. The root of the tree is the main goal to be proved, its children are the formulas that are used to prove this. Each node may have one, two, or no children depending on the number of formulas that the inference rule uses to derive the line. The

leaf nodes of the tree are those lines that are true without the need to combine other lines to prove it. (These are lines such as assumptions and premises.) Since some lines are used in the proof of several other lines, nodes in this explanation tree may have more than one parent.

The third dimension that can change in the explanation of natural deduction proofs is the *level* of the explanation. At one end of this dimension every line in the proof is explained. But, it is possible to imagine explanations where not every detail is given. In most proofs there are only a few major subgoals that need to be solved to justify or explain the top level goal. A very high level of explanation would just refer to them, and would not further describe how they were derived. With this type of explanation, the proof of certain of the subgoals would not be explained. They would be just stated as being provable and then used to explain the higher level goals. A natural place for this "cut-off" in explanation to occur would be to set a depth of embedding of subgoals after which they would not be explained. This "cut-off" could also eliminate some lines from the explanation that are above the cut-off depth, if it should turn out that they were only used in the proof of one of these subgoals. The subgoals at the cut-off depth will now become pseudo leaf nodes in the explanation tree. They will be treated as leaf nodes since their descendants will not be explained. The explanation can proceed normally but treat these nodes as leaves and call a function to explain them as "provable in further steps". Since the level at which cut-off can take place can vary there are several different explanations that can be generated, even while keeping all other dimensions constant.

As we have just discussed the cut-off mechanism allows for explanations where some subproofs are not explained. Sometimes one might wish to go back and examine the proof of just this subgoal and not the complete proof. Such an explanation would be done starting at the subgoal rather than at the first line in the proof. The explanation can

begin at any show line because each subproof is independent from the main proof and can be explained in isolation.

The fourth dimension where explanations of natural deduction proof can vary has to do with the *method* used to produce the explanation. The different methods of explanation come from how the explanation tree is traversed to generate the explanation. The traversal can be done either top-down or bottom-up. A third possibility is to mix the two and explain separate parts of the proof using the different schemes.

The first method of explanation is called top-down. This method corresponds to a pre-order traversal of the proof tree. At each node of the tree this style of explanation does several things. First, the current line's formula is stated as being what is to be proved. Secondly, the subgoals that are needed to justify this line are stated. After that, these subgoals are explained by recursively calling the top-down explanation function. The recursion is completed when an assumption or a premise is explained, since they have no more children in the explanation tree. Once the explanations of the subgoals has returned, the function may have a concluding sentence to tell how they are combined to derive the main goal. This step may be omitted for lines with no subgoals or those who are adequately explained in the first step.

The second method that can be used is called bottom-up, which is a post-order traversal of the proof tree. This approach begins the explanation by traversing down to the leaf nodes of the proof tree. At this point, those leaf-node lines are explained. This is simple since they are either assumptions or premises. An interior node of the tree is explained only when all its descendants are completely explained. The discourse continues, progressively expanding on those lines that have already been explained. The last node, or line, to be explained is the main goal of the proof. As can be seen, this type of explanation builds up from the simple to explain lines to the more complex.

The final variation on these methods of explanation of natural deduction proofs is a combination of the previous two; this is called a mixed type of explanation. The mixed explanation can begin either with top-down or bottom-up. Then several switching points can be set where the explanation will change from top-down to bottom-up or vice versa. These switching points correspond to a depth in the proof. All the subgoals at the same depth will be explained using the same strategy. (Either all bottom-up or all top-down.) This mixed technique can lead to many different types of explanations by varying the number of switches in method, and varying the levels at which these switches take place.

The following sections describe the four dimensions and how they were implemented. A more formal statement of the algorithm is given in chapter 5.

3.2 Explanations of the Formulas

3.2.1 Explanation of the Inference Rules

The formulas within each line of the natural deduction proof can be explained in several different ways. The first is not to explain or expand on the formulas, but rather just to use symbols in the explanation of the line in the proof. At this level of abstraction, the only thing that will be explained is the inference rule used to derive the line; and this means to explain how the inference rule was used to get the line in question. Hence, such an explanation only mentions what other formulas or lines are used. The proof found in figure 3 will be used as an example throughout the section of explanations of the formulas.

This level of explanation is called the inference rule level, and is quite simple. When a line is to be explained, a function that corresponds to the inference rule in the

INPUT:

Premise: $(\forall x_0)(BIRD(x_0) \rightarrow FEATHERS(x_0))$

Goal: $((BIRD(a_0) \& CHIRPS(a_0)) \rightarrow FEATHERS(a_0))$

OUTPUT:

1	*show $((BIRD(a_0) \& CHIRPS(a_0)) \rightarrow FEATHERS(a_0))$	2,3,IMP INT
2	| $(BIRD(a_0) \& CHIRPS(a_0))$	ASSUME
3	| *show $FEATHERS(a_0)$	4,8,CONTRA
4	| | $\neg FEATHERS(a_0)$	ASSUME
5	| | $BIRD(a_0)$	2,S
6	| | $(\forall x_0)(BIRD(x_0) \rightarrow FEATHERS(x_0))$	PREM
7	| | $(BIRD(a_0) \rightarrow FEATHERS(a_0))$	6,UI
8	| | $FEATHERS(a_0)$	7,5,MP

real		usersystem	
0.03	0.03	0.00	(sec.)

lines discarded: 1

300.00 lines/second (user time)

Figure 3: Example proof 1

line is called to do the explanation. In the example proof, line 1 would be explained by a function dedicated to explaining the introduction of implication. This function does several things to produce an explanation. First, it constructs the explanations for the formulas that will be used. In line 1 the formulas required are: $((BIRD(a_0) \& CHIRPS(a_0)) \rightarrow FEATHERS(a_0))$, $(BIRD(a_0) \& CHIRPS(a_0))$, and $FEATHERS(a_0)$. This would be the formula on the current line and the formulas from those lines that were used to derive it. At this point these formulas are not expanded, and thus the symbols are used

instead of an explanation. After this a random choice from several phrases to explain this particular inference rule is made. An example explanation for line 1 would be: "Since we assumed $(BIRD(a_0)\&CHIRPS(a_0))$ and then $FEATHERS(a_0)$ followed from that assumption then the conditional $((BIRD(a_0)\&CHIRPS(a_0))\rightarrow FEATHERS(a_0))$ can be derived." The formulas obtained previously are used to make these explanation phrases specific to the particular line. If the formulas were explained in sentences immediately previous to the current explanation, reference can be made to them and the formula need not be restated.

There are some differences between the explanation functions for top-down and those for bottom-up. The functions for top-down may have two places to choose a explanation phrase. First, there is a phrase selected to explain what must be proved to derive the current line, and then the recursive calls to explain the subgoals. After this, a second phrase may be used to tell how the subgoals are combined to form the current line. With the bottom-up method, only a single explanation phrase needs to be chosen, since the subgoals must have been explained before this line. Most often, the subgoals were explained immediately before and so the reference back to these sentences can be made instead of restating the formulas. There are also a few functions that can be used to explain an inference rule the same way whether using the top-down or bottom-up method. These correspond to inference rules like ASSUME and PREM that do not depend on any other lines for their proof. One other special function used by both is to explain that the line is provable in further steps. This function is used when cut-off is taking place in the explanation.

There are several inference rules whose explanation is usually not included in the complete explanation. The reason for this is that these rules can be easily inferred by the reader and are therefore not necessary to the explanation. One such inference rule is changing a biconditional to a conditional; another rule is conjunction simplification.

These are such simple rules that their use is obvious and need not be explained. If desired, these line may be included in the explanation of the proof, but they are usually left out.

When explaining an inference rule there are several stylistic varients that can be chosen from. For example, when explaining the inference rule IMP INT there are five phrases which can be chosen, these are given below. In these phrases T_1 and T_2 are the explanations of the formulas used by the inference rule and T_3 is the formula that is being explained.

1 We assumed T_1 and then proved that T_2 followed from that assumption therefore we get T_3.

2 T_2 followed by assuming T_1 and so we have proved T_3.

3 We now have T_3 because we assumed T_1 and then derived that T_2 followed from that assumption.

4 We proved that T_2 followed from assuming it is true that T_1, so, we can combiine this by a conditional to get T_3.

5 Since we assumed T_1 and then T_2 followed from that assumption then the conditional T_3 can be derived.

The choices for the other inference rules are similar in nature.

3.2.2 Explanation of the Connectives

The first level of explanation of a natural deduction proof was to simply explain the inference rules used. The formulas that occurred in the lines of the proof were left as they were in the proof. But the connectives that are found in the formulas can be easily translated into English phrases. Most connectives have an obvious translation such as

'and', 'or', 'if-then', 'for all', etc. These phrases can be used to expand on the explanation of the inference rules by further explaining the formulas that the rules work with. In this type of explanation the connectives are translated to English phrases and the predicates and constants used in the formulas are left as they were (as uninterpreted symbols). This is called explanation to the level of the connectives.

When a line in the proof is to be explained, the correct function (corresponding to the inference rule) is called. Then a call is made to explain the formulas that are to be used in the explanation of this line. When just the inference rules were explained this last type of call returned the formula in its uninterpreted form. But now that the connectives are to be explained, some further computation must be done to return the formula as a partially interpreted phrase. Each formula has a main connective; this connective can split the formula into one or two subformulas depending on the type of connective. For example, the main connective for the formula in line 1 is $\rightarrow$, implication, and the two subformulas are $(BIRD(a_0)\&CHIRPS(a_0))$ and $FEATHERS(a_0)$. To start the explanation of the formula, a function to explain the main connective is called. This function then recursively calls for the explanation of the subformulas. The recursion ends when a subformula has no main connective. In other words we have recursed down to a simple predicate. When the explanation of the subformulas has returned a phrase, a random choice is made from a set of explanatory phrases for this connective. Using the example above, the first subformula could be returned as the phrase "$BIRD(a_0)$ and $CHIRPS(a_0)$" and the second subformula will be returned as it was since it had no other connective. The subformula's explanation is used with the connective phrase to explain the complete formula. The implication explanation function can then return the phrase "($BIRD(a_0)$ and $CHIRPS(a_0)$) implies ($FEATHERS(a_0)$)". This complete explanation is now sent back to be used in the function that explains the inference rule where the phrase representing the

formula will be used instead of the formula itself.

There is one main problem with making the formula phrase easily understandable, which has to do with understanding the scope of the English connecting phrases. In large and complicated formulas there may be many connectives. The subformulas that they connect, their scope of reference, is made clear by the use of brackets in the uninterpreted formulas. When the formula is translated into an English phrase the brackets are usually removed. This causes a problem in determining what a certain connective phrase contains in its scope. To help reduce these ambiguities, some of the brackets have been reinserted into the explanation. This makes clear what each connective phrase is joining together.

The table below gives some of the stylistic varients for the various connectives. T_1 and T_2 are the left and right sub-formulas respectively.

Stylistic Varients for Main Connectives

Connective	Stylistic Varients		
$\neg$	it is false that T_2	T_2 is false	T_2 is not true
$\rightarrow$	if T_1 then T_2	T_2 follows from T_1	T_1 implies T_2
&	both T_1 and T_2		T_1 and T_2
$\vee$	either T_1 or T_2		T_1 or T_2
$\leftrightarrow$	we get T_1 if and only if we have T_2		T_1 follows from T_2 and vice versa
$\forall x_1$	for every constant x_1 we get T_2		for each constant x_1, T_2
$\exists x_1$	for at least one constant x_1, T_2		T_2 is true for at least one constant x_1

3.2.3 Explanation of the Predicates

Much of the time in a natural deduction proof the abstract predicate symbols that are used represent some relation in the real world, especially in computational applications. This information can be used as another refinement of the previous types of explanation. The inference rules and the connectives in the formulas can be explained as previously described; but now the predicates and constants that are used in the formulas can also be expanded into English phrases to make the explanation even more readable. This would lead to an explanation that is completely in natural language and has none of the symbols that were in the original proof. This is called explanation to the level of the predicates.

To give this type of explanation some additional information is needed. This is found in a "lexical information file" for the specific proof that is to be explained. This file includes information on the predicate and constant symbols that occur in the proof. For each predicate there are several pieces of lexical information. First, we state the number of arguments and prepositions for each. Secondly, we give several phrases for the predicate, such as the positive active, negative active, positive passive, and negative passive phrases. (Not all of these types of phrases need be given for each predicate.) A plural form of these phrases may also be given. Each constant that appears in the proof also is correlated with some information. The external form is given, for example, that a_0 represents tweety; further we state whether this is a singular or plural constant, and we give some pronouns that can be used for the constant. (A list is kept of all the constants in a single proof.) Some more general information is also included. Some phrases are given to represent universal quantification, eg. 'everyone', 'all birds' etc. Similar phrases are also used for existential quantification.

After all the lexical information is read in, the explanation may begin as it did before. When a formula is to be explained there is some in depth computation that must

take place to generate an English language phrase. There is a separate explanation function for each type of connective that can appear in a formula. This is a different set of functions than that of the connective explanation due to some special considerations made for the predicates being explained. The most important of these functions is the one to explain a formula that has no main connective. At this point, the formula will be a simple predicate. Other important connective explanation functions in the predicate level of explanation are those for universal quantification, existential quantification, negation, and double implication.

The key to predicate explanation is the function that translates a specific instance of a predicate into a natural language phrase. The first thing that is done is to find the structure that holds all the information for the particular predicate in question. To find this predicate a search is done of the linked list containing the predicate information. This process also binds the arguments in the predicate to their current values. This is done through a pointer from the argument structure to the correct constant in the constant list. Secondly, the function splits into different explanation pieces depending on the number of arguments that the predicate has. In the one argument case the function first checks to see if the negative form must be used. A flag is used here that may be set by the function to explain the connective for negation (Explained later). For example, while explaining the formula in line 1 the predicate instance $BIRD(a_0)$ has one argument and the positive form is to be used. The next choice is to decide whether to use the singular or plural form. The plural form will be used if the argument is a plural constant or if it is a variable that was universally instantiated. A check is also made to see whether the argument phrase can be substituted by a pronoun by keeping track of the last constant that was stated. The final phrase is generated by attaching the external forms for the arguments with the phrase for the predicate. In the example the final phrase produced is "tweety is a

bird". With a predicate that has two arguments another check is made before the creation of the final phrase. This is to see if both arguments are the same, for example $P(r_1,r_1)$. With this type of predicate the second argument can be replaced with a phrase such as "itself" or "himself". The generation of the phrase continues as before by choosing negative or positive form, passive or active versions of the phrase, plural or singular, and also whether either of the arguments can be replaced by a pronoun.

The formula explanation functions for universal and existential quantification connectives are very similar. We read from the lexical information file a set of constants and their translation to be used in the explanation. The universal and existential quantifiers give a variable a phrase for a particular formula. The function creates an entry for the variable in the constant list, since it will be used as a constant after it has a phrase associated with it. First the function, either universal or existential, will choose from a set of phrases that were read in earlier. For example, "all birds' or 'everyone' for universal and 'some bird' or 'someone' for the existential quantification. The chosen phrase will be the translation for the particular variable in this formula. This phrase will be in the entry for this variable in the constant list. In other words, the universal and existential quantification functions binds a phrase to the variable in question. Using the formula in line 6 when the universal instantiation connective is processed an entry in the constant list is produced for the variable x_0 and the phrase "every animal" is associated with it. The last thing for these functions to do is to call the explanation routine with the right subformula as its argument.

The second connective explanation function for which there is a significant difference when we add predicate explanation is the function for negation. When the connectives were the only thing to be explained the function would just produce explanations such as "X is false" or "X is not true" where X is the subformula that is negated.

When the predicates are also explained this must be changed when the subformula is one without a connective, in other words a simple predicate as in the formula of line 4. The negative phrases for the predicates were given in the lexical information file, and thus the negation function need not generate them itself but rather it can merely call the predicate explanation function with the subformula and a flag to indicate that the negative form should be used. Therefore the explanation for $\neg FEATHERS(a_0)$ is "tweety does not have feathers" rather than "tweety has feathers is false". When the subformula that is negated is more complex the same type of explanation as before is done.

Another function that differs from simple connective explanation when the predicates are included is that for the biconditional or double implication. When double implication, works with variables it acts as a constraint as to what the variable may be. This can lead to more complex sentences than just using "if and only if" type phrases. If the subformulas are predicates or negation of predicates, and the object of the first predicate matches the subject of the second, this situation can be exploited. The connective can be seen as a restriction on a variable and thus can be expressed as a relative clause such as "everyone who" or "all and only those men who".

The explanation of the formula in line 1 of figure 3 to the level of the predicates could be " Tweety is a bird and it chirps implies that it has feathers".

3.3 Levels of Explanation

The implementation of the cut-off mechanism in EXPLAIN is straight forward. A level at which the cut-off is to be performed at is set at the start of the explanation. If there is to be no cut-off this level is set to a depth deeper than any lines in the proof. When a line is to be explained it is first checked as to whether it is a subgoal or not, since a cut-off only occurs at subgoals and not at simple lines in the proof. Lines at the cut-off

depth that are not subgoals are explained since the lines on which it depends must be *antecedent* and thus no deeper in depth than the current line. Those simple line that are deeper than the cut-off won't be explained since they are used to prove a subgoal that must be at the cut-off level or deeper. Some lines that are above the cut-off may also be eliminated from the explanation because they are used exclusively to prove a cut-off subgoal. If the current line is a subgoal its level is compared to the cut-off depth. If the level of the line is equal to the cut-off depth then a function is called to state that the line is provable by some further steps that will not be explained. The line is then marked as explained and the explanation continues normally. This cut-off method is implemented the same in both the top-down and bottom-up methods and so will also work in combination with a mixed explanation as well as the two simple methods.

The facility to begin an explanation at any show line in the proof is also easy to implement. The explanation functions are recursive and so the only argument that is passed to them is the line number to explain. To start the explanation at a different point the initial line number passed to the function will be the line number of a subgoal rather than 1 which is the line number of the main goal.

3.4 Methods for Explanation

3.4.1 Bottom-up Method

One way to explain a natural deduction proof is to traverse the proof tree in a bottom-up manner. In EXPLAIN there is a straight forward way to achieve this. When a line is to be explained first it is checked to see if it has any subgoals; if so they are explained by a recursive call. After they are explained, or if there are no subgoals, we choose the correct explanation function as determined by the inference rule for the

current line. This is the function which then does the explanation of this individual line. It might not always be a good idea to use the strict bottom-up explanation method all the time, for there are certain cases where we would like to deviate from this to make the explanation easier to understand.

In the first place, the so-called proof tree is not a true tree: there are some nodes that may appear in the tree at more than one place. This means that the tree should be seen as a more general graph. Once a node has been explained it would not be desirable to do the complete explanation of the same node again even if it appears once again in the tree. So, we keep track of which lines, or nodes, have been explained so far. Using this, if a line is reached that has already been explained we can eliminate the presentation of another complete explanation. Instead the line can be simply stated again and reference made back to the previous explanation. For example, "We proved X previously.".

In the second place, entries in the proof tree are not in the same order that the lines of the proof were constructed. This can lead to some problems when explaining the proof by using the order given by the tree. The biggest problem is with the instantiation of variables. A constant can be introduced into the proof in two ways, in addition to being in a premise. First, it could come about by instantiation of an existential quantifier, and second by the instantiation of a universal quantifier. The rule justifying use of a existential quantifier to introduce a constant is much more restrictive than the rule justifying the use of a universal quantifier, in that the constant must be *entirely* new to the proof, whereas when a universal quantifier is used *any* variable new or old may be introduced. It is here that a problem may occur, at least in some variants of giving an explanation. For, it is required (both in the proof and in any adequate recounting of the proof in English) that when a constant is first introduced by existential instantiation and then later referred to in a universal instantiation, that the existential introduction be described

first. But a bottom-up explanation of the proof may not preserve this order that it was performed in during the proof. Therefore whenever a universal instantiation takes place, we check to see whether a constant is introduced by an existential quantifier earlier in the proof, but not the explanation. If this is so, we insert a side explanation so the existential introduction of the constant comes first in the explanation, as it should logically. An example of this problem and its solution is found in section 3.4.3.

3.4.2 Top-down Method

The second method for explaining a natural deduction proof is called top-down. This method does the explanation by doing a pre-order traversal of the proof tree. This means that when a node is reached it is explained and then its children are explained.

The algorithm used to do the top-down explanation is very similar to what is done for the bottom-up method. When a line is to be explained the first thing done is to decide which explanation function must be called for this particular line. There is a separate function for each inference rule that could be associated with a line. Within the called function the current line is explained and then there is a recursive call to the explanation routine to explain the subgoals (if there are any). After this is done, a sentence may be added to describe how the subgoals are combined to produce the current line. As in the bottom-up method , it is not possible to follow this algorithm strictly. As before lines that have already been explained are kept track of so that there is no unnecessary re-explanation of lines. And again as before, the top-down method of explanation can also lead to the same problem concerning existential and universal instantiation. The top-down strategy deals with this problem in the same manner as did the bottom-up strategy.

INPUT:

Goal: $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$

OUTPUT:

1	*show $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$	7,9,CONTRA
2	| $(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$	ASSUME
3	| $(B(r_1)\&(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$	2,EI
4	| $(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1))$	3,S
5	| $(P(r_1,r_1)\leftrightarrow\neg P(r_1,r_1))$	4,UI
6	| $(\neg P(r_1,r_1)\rightarrow P(r_1,r_1))$	5,BC
7	| *show $P(r_1,r_1)$	6,8,MP
8	| | $\neg P(r_1,r_1)$	ASSUME
9	| $\neg P(r_1,r_1)$	5,7,EQ

real		usersystem	
0.04	0.04	0.00	(sec.)

lines discarded: 2

275.00 lines/second (user time)

Figure 4: Example proof 2

3.4.3 Sample Proof and the Order of Explanation

If we look at figure 4 the order in which the lines are explained using the bottom-up method would be 2 3 4 5 6 8 7 9 1. The main goal, line 1, is explained last, while simple lines such as the assumption in line 2 are explained earlier in the proof. This proof requires no deviations from the normal bottom-up explanation procedure.

Using the same example, a pure top-down explanation would yield a order of 1 7 6 5 4 3 2 8 9. This method produces one of the problems mentioned earlier. At line 3 the constant r_1 is introduced by existential instantiation. In line 5 the same constant is presented by universal instantiation. The proof is correct since line 5 follows line 3 but the top-down explanation does not preserve this order. The explanation may appear wrong since it is not valid to introduce a constant by existential instantiation that already appears in the proof. Therefore, a side explanation must be produced to insure the order of these two lines is not violated. The resulting explanation order would now be 1 7 6 3 2 5 4 8 9. The main goal is explained first rather than leaving it to the end as is done with a pure bottom-up explanation.

3.4.4 Mixed Method

There are advantages and disadvantages to both of top-down and bottom-up methods of explanation. To get the best of both, a third method of explanation might be tried, where some parts of the proof are explained top-down and others bottom-up. This method will be called a mixed method explanation. Changing from one type of explanation to another is done by using the depth of the embedded subgoals in the proof. A switch in explanation is only done with subgoals. Lines that are not subgoals will not change the method of explanation regardless of their depth. Methods used can be switched many times throughout the explanation of a proof. For example, an explanation could start top-down and then at depth two change to bottom-up and then at depth four revert to top-down and so on. In such an example, all subgoals at the top level would be explained top-down; subgoals at depth two and three would be explained using the bottom-up method; and any subgoals at a depth of four or greater would be explained by using the top-down method.

EXPLAIN uses a simple way to implement this switching back and forth between the two methods of explanation. There are a series of marks set to indicate at which level to switch the explanation method. There may be several depths at which a switch is to take place. When a line is to be explained it is first checked as to whether it is a subgoal or a simple line of the proof. If the current line is a subgoal then it is possible that the method of explanation could change. The depth of the current line is checked against the current switching level, and if they match a switch of the explanation method is to be done. Thereafter, the current switching level is changed to the next depth at which a switch is to be made. If there are to be no more switches the current switching level is set to some level out of range with the proof. Next, the other method (either top-down or bottom-up) is called recursively with the line that caused the switch as the goal. When the explanation of this line returns the current switching level is reset to its previous value. The explanation of the proof will then proceed normally.

4. Examples

4.1 Examples of Proof Explanations

Many systems which produce explanations for natural deduction proofs concentrate on small proofs, of the kind which can be explained to even naive users. But, in most cases the user of such a system would not be completely naive, when it comes to logic, and would be able to understand these proofs without an explanation. The proofs which need to be explained are those which are large and complicated. The explanations produced here are not meant to be understandable by the ordinary person with no knowledge of logic what so ever. The purpose is to not force the users of the system to be a logician, rather those with some understanding of logic (like a computer programmer) can now comprehend the proofs without having to consult a logician for the explanation. For long proofs the uninitiated (logically) would have a hard time following even the best worded explanations. The user must have some desire and some logical ability to be able to follow the explanation generated. In most cases the logical proof can also be produced and the explanation can be used to guide the reader through the proof.

At the end of each explanation the time for the explanation to be produced is printed.

The first set of examples all deal with the following proof generated by THINKER. The first example is of a top-down explanation. The second demonstrates the bottom-up method. Thirdly, an example is given to show an explanation to the level of the connectives. And finally a complete English language proof is presented (predicate level of explanation). The output of the EXPLAIN facility is reproduced here exactly as it was

produced by the program, and has not been edited in any way.

===test60===

INPUT:

Goal: $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$

OUTPUT:

1	*show* $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$	7,9,CONTRA
2	| $(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$	ASSUME
3	| $(B(r_1)\&(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$	2,EI
4	| $(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1))$	3,S
5	| $(P(r_1,r_1)\leftrightarrow\neg P(r_1,r_1))$	4,UI
6	| $(\neg P(r_1,r_1)\rightarrow P(r_1,r_1))$	5,BC
7	| *show* $P(r_1,r_1)$	6,8,MP
8	| | $\neg P(r_1,r_1)$	ASSUME
9	| $\neg P(r_1,r_1)$	5,7,EQ

real	user	system	
0.03	0.02	0.01	(sec.)

lines discarded: 2
550.00 lines/second (user time)

THE EXPLANATION IN BOTTOM-UP FASHION

The goal of the proof is to prove $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$. We assume $(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$. To get $(B(r_1)\&(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$ we substitute the new constant r_1 for y_1 in $(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$. We set the variable x_1 to equal r_1 in $(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1))$. Assume $\neg P(r_1,r_1)$. $(\neg P(r_1,r_1)\rightarrow P(r_1,r_1))$ was proven earlier. And using the last two formulas, we can get $P(r_1,r_1)$ since the antecedent of the conditional is true. We previously proved $(P(r_1,r_1)\leftrightarrow\neg P(r_1,r_1))$. Using the last two formulas, since one side is true and the equality is true we can derive the consequent of $\neg P(r_1,r_1)$. We have generated a contradiction $(P(r_1,r_1))$ and $(\neg P(r_1,r_1))$ and therefore we can conclude that $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$.

real	user	system	
0.02	0.01	0.01	(sec.)

THE EXPLANATION IN TOP-DOWN FASHION

To prove that $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$ is true we want to generate a contradiction. We can get $P(r_1,r_1)$ if we can prove both $\neg P(r_1,r_1)$, and $(\neg P(r_1,r_1)\rightarrow P(r_1,r_1))$. To show $(B(r_1)\&(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$ is true we have to prove $(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$ and then we can instantiate the variable y_1 with the new constant r_1. Assume $(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$. . To get $(P(r_1,r_1)\leftrightarrow\neg P(r_1,r_1))$ we must prove that $(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1))$ is true and then substitute r_1 for x_1. As we saw earlier $(B(r_1)\&(\forall x_1)(P(r_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$ is true. Assume $\neg P(r_1,r_1)$. We can now get $P(r_1,r_1)$ because we assumed $\neg P(r_1,r_1)$, and proved $(\neg P(r_1,r_1)\rightarrow P(r_1,r_1))$ and so we can get the conclusion of the conditional statement. In order to prove the statement $\neg P(r_1,r_1)$ we will show that $(P(r_1,r_1)\leftrightarrow\neg P(r_1,r_1))$ and $P(r_1,r_1)$ are both true. We previously proved $(P(r_1,r_1)\leftrightarrow\neg P(r_1,r_1))$. $P(r_1,r_1)$ was proven earlier. Because the biconditional, $(P(r_1,r_1)\leftrightarrow\neg P(r_1,r_1))$, was proven and one side is true $(P(r_1,r_1))$ the other side $(\neg P(r_1,r_1))$ must also be true. Since we have generated a contradiction $(P(r_1,r_1))$ and $(\neg P(r_1,r_1))$ we can conclude $\neg(\exists y_1)(B(y_1)\&(\forall x_1)(P(y_1,x_1)\leftrightarrow\neg P(x_1,x_1)))$.

real	user	system	
0.02	0.01	0.01	(sec.)

EXPLANATION AT THE LEVEL OF THE CONNECTIVES

The goal of the proof is to prove (there is no constant y_1 that gives us (both $B(y_1)$ and (for every constant x_1, we get ($P(y_1,x_1)$ follows from ($P(x_1,x_1)$ is false) and vice versa)))). Now, let's suppose (there is a constant y_1 that gives us ($B(y_1)$ and (for each constant x_1, (if we have $P(y_1,x_1)$ we get ($P(x_1,x_1)$ is not true) and vice versa)))). We replace y_1 by a new constant r_1 in (both $B(r_1)$ and (for any constant x_1, (we get, $P(r_1,x_1)$, if and only if we have, ($P(x_1,x_1)$ is false)))). We get (if we have $P(r_1,r_1)$ we get ($P(r_1,r_1)$ is false) and vice versa) by substituting r_1 for x_1 in the statement (for any constant x_1, (we get, $P(r_1,x_1)$, if and only if we have, ($P(x_1,x_1)$ is not true))). Now, let's suppose ($P(r_1,r_1)$ is false). As we saw earlier (if ($P(r_1,r_1)$ is false), then $P(r_1,r_1)$) is true. And using the last two formulas, since both the antecedent and the conditional are true we can derive the consequent of $P(r_1,r_1)$. Earlier in the proof we proved ($P(r_1,r_1)$ follows from ($P(r_1,r_1)$ is not true) and vice versa). From the previous two, we can get ($P(r_1,r_1)$ is not true) since one side of the equality is true. We have generated a contradiction ($P(r_1,r_1)$) and (($P(r_1,r_1)$ is not true)) and therefore we can conclude that (there is no constant y_1 that gives us (both $B(y_1)$ and (for any constant x_1, (we get, $P(y_1,x_1)$, if and only if we have, ($P(x_1,x_1)$ is false))))).

real	user	system	
0.03	0.01	0.02	(sec.)

EXPLANATION AT THE LEVEL OF THE PREDICATES

The main goal is to prove it is false that somebody is a barber and he shaves all and only those people who don't shave themselves. Now, let's suppose someone is a barber and he shaves all and only those people who don't shave themselves. Let's select somebody arbitrarily and call him Fred. Fred is a barber and he shaves all and only those people who don't shave themselves would follow from the fact that somebody does this and Fred is arbitrary. We can get that Fred shaves himself if and only if Fred doesn't shave himself by particularizing the earlier claim that Fred shaves all and only those people who don't shave themselves. Now, let's suppose Fred doesn't shave himself. It was proved before that Fred is shaved by himself follows from Fred doesn't shave himself. And from these two we derive that Fred shaves himself because the antecedent being true implies that the consequent is also true. We proved earlier in the proof that Fred shaves himself if and only if Fred doesn't shave himself. It follows from these two since one side is true and the equality is true we can derive the consequent of Fred doesn't shave himself. We have generated a contradiction (Fred is shaved by himself) and (Fred doesn't shave himself) and therefore we can conclude that it is not true that somebody is a barber and he shaves all and only those people who don't shave themselves.

real	user	system	
0.23	0.03	0.02	(sec.)

The next two examples show first a connective level explanation and second a predicate level explanation.

===test7===

INPUT:

Premise: $(\forall x_0)BIRD(x_0)\rightarrow FEATHERS(x_0)$

Goal: $((BIRD(a_0)\&CHIRPS(a_0))\rightarrow FEATHERS(a_0))$

OUTPUT:

1	$*show\ ((BIRD(a_0)\&CHIRPS(a_0))\rightarrow FEATHERS(a_0))$	2,3,IMP INT
2	$\mid (BIRD(a_0)\&CHIRPS(a_0))$	ASSUME
3	$\mid *show\ FEATHERS(a_0)$	4,8,CONTRA
4	$\mid\mid \neg FEATHERS(a_0)$	ASSUME
5	$\mid\mid BIRD(a_0)$	2,S
6	$\mid\mid (\forall x_0)BIRD(x_0)\rightarrow FEATHERS(x_0))$	PREM
7	$\mid\mid (BIRD(a_0)\rightarrow FEATHERS(a_0))$	6,UI
8	$\mid\mid FEATHERS(a_0)$	7,5,MP

real	user	system	
0.02	0.02	0.00	(sec.)

lines discarded: 1
450.00 lines/second (user time)

THE EXPLANATION IN TOP-DOWN FASHION TO THE LEVEL OF THE CONNECTIVES

To prove (if (both $BIRD(a_0)$ and $CHIRPS(a_0)$), then $FEATHERS(a_0)$) we shall assume ($BIRD(a_0)$ and $CHIRPS(a_0)$) and then try to prove that $FEATHERS(a_0)$ follows from this assumption. To prove $FEATHERS(a_0)$ is true we will assume that it is false and then generate a contradiction from this assumption. To prove $FEATHERS(a_0)$ we want to prove that it can be derived from $BIRD(a_0)$, and (if $BIRD(a_0)$, then $FEATHERS(a_0)$) both being true. We want to get (if $BIRD(a_0)$, then $FEATHERS(a_0)$). To do this we prove (for all constants represented by x_0, ($BIRD(x_0)$ implies $FEATHERS(x_0)$)) and then substitute a_0 for x_0. The statement (for each constant x_0, (if we have, $BIRD(x_0)$, then, $FEATHERS(x_0)$ follows)) is known to be true. We assume ($BIRD(a_0)$ and $CHIRPS(a_0)$). Since we proved $BIRD(a_0)$, and (if $BIRD(a_0)$, then $FEATHERS(a_0)$) we can get $FEATHERS(a_0)$ because the first part of the conditional is true therefore the second part is also true. Our assumption ($FEATHERS(a_0)$ is not true) is false since we have a contradiction (($FEATHERS(a_0)$ is false)) and ($FEATHERS(a_0)$) , therefore we get $FEATHERS(a_0)$. So (if (both $BIRD(a_0)$ and $CHIRPS(a_0)$), then $FEATHERS(a_0)$) is true because we assumed ($BIRD(a_0)$ and $CHIRPS(a_0)$) and then showed that $FEATHERS(a_0)$ followed from that assumption.

real	user	system	
0.02	0.02	0.00	(sec.)

THE EXPLANATION IN TOP-DOWN FASHION TO THE LEVEL OF THE PREDICATES

To prove tweety is a bird and it chirps implies that it has feathers we shall assume it is a bird and it chirps and then try to prove that it has feathers follows from this assumption. We will prove tweety has feathers by generating a contradiction using the assumption that tweety has feathers is false. To prove tweety has feathers we want to prove that it can be derived from tweety is a bird, and if we have that it is a bird, then, it has feathers follows both being true. To get that tweety is a bird implies that it has feathers, we must prove it is true that every animal is a bird implies that they have feathers and then use tweety to make it more specific. We are told it is true that if every animal is a bird, then it follows that they have feathers. Suppose tweety is a bird and it chirps. Since we proved tweety is a bird, and if we have that it is a bird, then, it has feathers follows we can get tweety has feathers because the first part of the conditional is true therefore the second part is also true. We have a contradiction (tweety does not have feathers) and (tweety has feathers) and so we must repeal our assumption tweety does not have feathers and thus tweety has feathers is true. So tweety is a bird and it chirps implies that it has feathers is true because we assumed it is a bird and it chirps and then showed that it has feathers followed from that assumption.

real	user	system	
0.25	0.00	0.03	(sec.)

The next two examples deal with the mixed type of explanation. The first is set so that the subgoals at levels 0 (the main subgoal) and 1 are explained top-down and those at level 2 or deeper are explained bottom-up. In the second example the methods are reversed.

====test5====

INPUT:

Premise: $(q \rightarrow r)$
Premise: $(r \rightarrow \neg(q \& p))$
Premise: $(p \rightarrow \neg(q \vee r))$
Goal: $(p \leftrightarrow q)$

OUTPUT:

1	*show $(p \leftrightarrow q)$	11,2,CB
2	| *show $(q \rightarrow p)$	3,4,IMP INT
3	| | q	ASSUME
4	| | *show p	5,10,CONTRA
5	| | | $\neg p$	ASSUME
6	| | | $(q \rightarrow r)$	PREM
7	| | | r	6,3,MP
8	| | | $(r \rightarrow \neg(q \& p))$	PREM
9	| | | $(q \& p)$	8,7,MP
10	| | | p	9,S
11	| *show $(p \rightarrow q)$	12,13,IMP INT
12	| | p	ASSUME
13	| | *show q	14,20,CONTRA
14	| | | $\neg q$	ASSUME
15	| | | $(p \rightarrow \neg(q \vee r))$	PREM
16	| | | $(q \vee r)$	15,12,MP
17	| | | r	16,14,MTP
18	| | | $(r \rightarrow \neg(q \& p))$	PREM
19	| | | $(q \& p)$	18,17,MP
20	| | | q	19,S

real	user	system	
0.03	0.02	0.01	(sec.)

lines discarded: 0
1000.00 lines/second (user time)

MIXED EXPLANATION BEGINNING IN TOP-DOWN FASHION

In order to prove $(p \leftrightarrow q)$ we have to show both $(p \rightarrow q)$ and $(q \rightarrow p)$. We want to prove $(p \rightarrow q)$. To do this we shall assume p and then show that q can then be derived.

Now, let's suppose $\neg q$. $(r \rightarrow (q \& p))$ is a fact. It is known that $(p \rightarrow (q \vee r))$ is true. We assume p. Following the above formula and the last premise, since both the antecedent and the conditional are true we can derive the consequent of $(q \vee r)$. We assumed $\neg q$ earlier in the proof. From these two one part of a true disjunction is false which means the other part must be true and we get r. We stated that $(r \rightarrow (q \& p))$ was a premise earlier. Using the last two formulas, we can get $(q \& p)$ since the antecedent of the conditional is true. We have both true (q) and false $(\neg q)$ of the same formula, which is a contradiction. Therefore, we must repeal our assumption $\neg q$ and we have proven q.

We can now conclude $(p \rightarrow q)$ since q was derived after assuming, p. To prove $(q \rightarrow p)$ we shall assume q and then try to prove that p follows from this assumption.

Suppose $\neg p$. It is a fact that $(r \rightarrow (q \& p))$ is true. $(q \rightarrow r)$ is a premise. Now, let's suppose q. Following the above formula and the last premise, since both the antecedent and the conditional are true we can derive the consequent of r. $(r \rightarrow (q \& p))$, was previously given as a premise. From the previous two, we can get $(q \& p)$ since the antecedent of the conditional is true. We get p because our original assumption $\neg p$ led us to a contradiction $(\neg p)$ and (p).

So $(q \rightarrow p)$ is true because we assumed q and then showed that p followed from that assumption. We now have $(p \leftrightarrow q)$ because we can combine the two formulas $(p \rightarrow q)$ and $(q \rightarrow p)$.

real	user	system	
0.02	0.00	0.02	(sec.)

MIXED EXPLANATION BEGINNING IN BOTTOM-UP FASHION

We want to prove $(p \leftrightarrow q)$. We assume p.

To prove q is true we will assume that it is false and then generate a contradiction from this assumption. We can get $(q \& p)$ if we can prove both r, and $(r \rightarrow (q \& p))$. We know $(r \rightarrow (q \& p))$ is a premise. In order to get r we must first show that $(q \vee r)$ and $\neg q$ are true. We want to get $(q \vee r)$ to do this we have to prove the two formulas p, and $(p \rightarrow (q \vee r))$. The statement $(p \rightarrow (q \vee r))$ is known to be true. Earlier in the proof we assumed p. It is possible to conclude $(q \vee r)$ because we assumed p, and proved $(p \rightarrow (q \vee r))$ therefore the second part of the conditional can be deduced. $\neg q$ is assumed. We can deduce r because the disjunction $(q \vee r)$ is true and we also have $\neg q$ so the other side must be true. We can now get $(q \& p)$ because we proved both r, and $(r \rightarrow (q \& p))$ and so we can get the conclusion of the conditional statement. Our assumption $\neg q$ is false since we have a contradiction $(\neg q)$ and (q), therefore we get q.

We assumed p and then proved that q followed from that assumption therefore we get $(p \rightarrow q)$. Now, let's suppose q.

We will prove p by generating a contradiction using the assumption that p is false. To prove $(q \& p)$ we want to prove that it can be derived from r, and $(r \rightarrow (q \& p))$ both being true. We know $(r \rightarrow (q \& p))$ is a premise. We want to get r to do this we have to prove the two formulas q, and $(q \rightarrow r)$. $(q \rightarrow r)$ is a fact. We assumed q earlier in the proof. It is possible to conclude r because we assumed q, and proved $(q \rightarrow r)$ therefore the second part of the conditional can be deduced. Since we proved r, and $(r \rightarrow (q \& p))$ we can get $(q \& p)$ because the first part of the conditional is true therefore the second part is also true. We have a contradiction $(\neg p)$ and (p) and so we must repeal our assumption $\neg p$ and thus p is true.

We proved that p followed from assuming it is true that q, so, we can combine this by a conditional to get $(q \rightarrow p)$. $(p \rightarrow q)$ was proven earlier. And from the previous two, since the each side implies the other they are equal and can be written as $(p \leftrightarrow q)$.

real	user	system	
0.04	0.01	0.03	(sec.)

The next three examples are different explanations of the proof displayed on the next page. The first displays mixing of explanation methods by having all subgoals at level 3 or greater explained top-down and those above bottom-up. The next mixed example shows how the explanation can switch several times. The top level goal is explained top-down, those at level 1 and 2 are done bottom-up and all subgoals at level 3 or more are explained top-down. Lastly, a high level example is given where subgoals at level 3 are not completely explained.

===test62===

INPUT:
Goal: $((p \to q) \leftrightarrow (\neg q \to \neg p))$

OUTPUT:

```
 1   *show  ((p →q )↔(¬q →¬p ))                    9,2,CB
 2   | *show  ((¬q →¬p )→(p →q ))                  3,4,IMP INT
 3   | | (¬q →¬p )                                  ASSUME
 4   | | *show  (p →q )                            5,6,IMP INT
 5   | | | p                                        ASSUME
 6   | | | *show  q                                 5,8,CONTRA
 7   | | | | ¬q                                     ASSUME
 8   | | | | ¬p                                     3,7,MP
 9   | *show  ((p →q )→(¬q →¬p ))                   10,11,IMP INT
10   | | (p →q )                                    ASSUME
11   | | *show  (¬q →¬p )                           12,13,IMP INT
12   | | | ¬q                                       ASSUME
13   | | | *show  ¬p                                12,15,CONTRA
14   | | | | p                                      ASSUME
15   | | | | q                                      10,14,MP
```

real	user	system	
0.03	0.03	0.00	(sec.)

lines discarded: 0
500.00 lines/second (user time)

MIXED EXPLANATION BEGINNING IN BOTTOM-UP FASHION

$((p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p))$ is what is to be proved. Assume $(p \rightarrow q)$. We will also assume $\neg q$.

To prove that $\neg p$ is true we want to generate a contradiction. We assumed $\neg q$ earlier. We can get q if we can prove both p, and $(p \rightarrow q)$. We assumed $(p \rightarrow q)$ earlier. Assume p. We can now get q because we assumed both p, and $(p \rightarrow q)$ and so we can get the conclusion of the conditional statement. Since we have generated a contradiction $(\neg q)$ and (q) we can conclude $\neg p$.

$\neg p$ followed by assuming $\neg q$ and so we have proven $(\neg q \rightarrow \neg p)$. We now have $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ because we assumed $(p \rightarrow q)$ and then derived that $(\neg q \rightarrow \neg p)$ followed from that assumption. We assume $(\neg q \rightarrow \neg p)$. We will also assume p.

We will prove q by generating a contradiction using the assumption that q is false. We want to get $\neg p$ to do this we have to prove the two formulas $\neg q$, and $(\neg q \rightarrow \neg p)$. We assumed $(\neg q \rightarrow \neg p)$ earlier in the proof. $\neg q$ is assumed. It is possible to conclude $\neg p$ because we assumed that $\neg q$, and $(\neg q \rightarrow \neg p)$ are true therefore the second part of the conditional can be deduced. We have a contradiction (p) and $(\neg p)$ and so we must repeal our assumption $\neg q$ and thus q is true.

We assumed p and then proved that q followed from that assumption therefore we get $(p \rightarrow q)$. We proved that $(p \rightarrow q)$ followed from assuming it is true that $(\neg q \rightarrow \neg p)$, so, we can combine this by a conditional to get $((\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q))$. We proved $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ earlier in the proof. And using the last two formulas, we get $((p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p))$ by joining them with a biconditional, or equality.

```
real     user     system
0.02     0.01     0.01     (sec.)
```

A MIXED EXPLANATION BEGINNING IN TOP-DOWN FASHION

To prove $((p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p))$ we must show that each implication $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ and $((\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q))$ are true.

We assume $(p \rightarrow q)$. We can also assume $\neg q$.

To prove that $\neg p$ is true we want to generate a contradiction. We assumed $\neg q$ earlier. To prove q we want to prove that it can be derived from p, and $(p \rightarrow q)$ both being true. We assumed $(p \rightarrow q)$ earlier in the proof. Now, let's suppose p. Since we assumed p, and $(p \rightarrow q)$ we can get q because the first part of the conditional is true therefore the second part is also true. Since we have generated a contradiction $(\neg q)$ and (q) we can conclude $\neg p$.

Since we assumed $\neg q$ and then $\neg p$ followed from that assumption then the conditional $(\neg q \rightarrow \neg p)$ can be derived. We proved that $(\neg q \rightarrow \neg p)$ followed from assuming it is true that $(p \rightarrow q)$, so, we can combine this by a conditional to get $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$.

Now, let's suppose $(\neg q \rightarrow \neg p)$. We can also assume p.

We will prove q by generating a contradiction using the assumption that q is false. We want to get $\neg p$ to do this we have to prove the two formulas $\neg q$, and $(\neg q \rightarrow \neg p)$. We assumed $(\neg q \rightarrow \neg p)$ earlier in the proof. Assume $\neg q$. It is possible to conclude $\neg p$ because we assumed that $\neg q$, and $(\neg q \rightarrow \neg p)$ are true therefore the second part of the conditional can be deduced. We have a contradiction (p) and $(\neg p)$ and so we must repeal our assumption $\neg q$ and thus q is true.

We now have $(p \rightarrow q)$ because we assumed p and then derived that q followed from that assumption. Since we assumed $(\neg q \rightarrow \neg p)$ and then $(p \rightarrow q)$ followed from that assumption then the conditional $((\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q))$ can be derived.

We know $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ and $((\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q))$ so we can combine them to get $((p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p))$.

real	user	system	
0.03	0.01	0.02	(sec.)

THE EXPLANATION IN BOTTOM-UP FASHION CUT OFF AT LEVEL 3

$((p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p))$ is what is to be proved. Assume $(p \rightarrow q)$. $\neg q$ can be assumed as well. We can prove $\neg p$ in some other steps. $\neg p$ followed by assuming $\neg q$ and so we have proven $(\neg q \rightarrow \neg p)$. We assumed $(p \rightarrow q)$ and then proved that $(\neg q \rightarrow \neg p)$ followed from that assumption therefore we get $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$. We assume $(\neg q \rightarrow \neg p)$. We will also assume p. We will leave out the details of proving that q is true. Since we assumed p and then q followed from that assumption then the conditional $(p \rightarrow q)$ can be derived. We assumed $(\neg q \rightarrow \neg p)$ and then proved that $(p \rightarrow q)$ followed from that assumption therefore we get $((\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q))$. As we saw earlier $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ is true. And from these two since the each side implies the other they are equal and can be written as $((p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p))$.

real	user	system	
0.01	0.01	0.00	(sec.)

The final set of examples all deal with a rather long proof. This is where a high level explanation may be more useful. The first example makes the "cut-off" at level 4 in top-down fashion and the second does the explanation up to level 5 bottom-up. The next two examples show how just a subgoal can be explained by starting the explanation of the proof at lines 30 and 39 respectively.

===test41===

INPUT:

Premise: $(Ax_0)(N(x_0)\rightarrow H(x_0))$
Premise: $(Ax_0)((W(x_0)\&H(x_0))\rightarrow B(x_0))$
Premise: $(Ax_0)(Ay_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$
Premise: $(Ay_0)(P(y_0,x_0)\rightarrow N(y_0))$
Premise: $(Ax_0)(Ey_0)(W(y_0)\&P(y_0,x_0))$
Goal: $(Ax_0)((Ay_0)(P(y_0,x_0)\rightarrow N(y_0))\rightarrow B(x_0))$

OUTPUT:

1	*show* $(\forall x_0)((\forall y_0)(P(y_0,x_0)\rightarrow N(y_0))\rightarrow B(x_0))$	2,UG
2	| *show* $((\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))\rightarrow B(r_1))$	3,4,IMP INT
3	| | $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$	ASSUME
4	| | *show* $B(r_1)$	16,28,CONTRA
5	| | | $\neg B(r_1)$	ASSUME
6	| | | $(\forall x_0)(N(x_0)\rightarrow H(x_0))$	PREM
7	| | | $(N(r_1)\rightarrow H(r_1))$	6,UI
8	| | | $(\forall x_0)((W(x_0)\&H(x_0))\rightarrow B(x_0))$	PREM
9	| | | $((W(r_1)\&H(r_1))\rightarrow B(r_1))$	8,UI
10	| | | $(\forall x_0)(\forall y_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$	PREM
11	| | | $(\forall y_0)((P(y_0,r_1)\&B(y_0))\rightarrow B(r_1))$	10,UI
12	| | | $(\forall x_0)(\exists y_0)(W(y_0)\&P(y_0,x_0))$	PREM
13	| | | $(\exists y_0)(W(y_0)\&P(y_0,r_1))$	12,UI
14	| | | $(P(r_2,r_1)\rightarrow N(r_2))$	3,UI
15	| | | $((P(r_2,r_1)\&B(r_2))\rightarrow B(r_1))$	11,UI
16	| | | $\neg(W(r_2)\&H(r_2))$	9,5,MT
17	| | | $(W(r_2)\&P(r_2,r_1))$	13,EI
18	| | | $\neg(P(r_2,r_1)\&B(r_2))$	15,5,MT
19	| | | $W(r_2)$	17,S
20	| | | $P(r_2,r_1)$	17,S
21	| | | $(N(r_2)\rightarrow H(r_2))$	6,UI
22	| | | $((W(r_2)\&H(r_2))\rightarrow B(r_2))$	8,UI
23	| | | $(P(r_2,r_1)\rightarrow N(r_2))$	3,UI
24	| | | $((P(r_2,r_1)\&B(r_2))\rightarrow B(r_1))$	11,UI
25	| | | $N(r_2)$	23,20,MP
26	| | | $\neg(P(r_2,r_1)\&B(r_2))$	24,5,MT
27	| | | $H(r_2)$	21,25,MP
28	| | | *show* $(W(r_2)\&H(r_2))$	36,29,ADJ
29	| | | | *show* $W(r_2)$	18,30,CONTRA
30	| | | | | *show* $(P(r_2,r_1)\&B(r_2))$	26,31,CONTRA
31	| | | | | | *show* $(P(r_2,r_1)\&B(r_2))$	32,20,ADJ
32	| | | | | | | *show* $B(r_2)$	34,35,CONTRA
33	| | | | | | | | $\neg B(r_2)$	ASSUME
34	| | | | | | | | $\neg(W(r_2)\&H(r_2))$	22,33,MT
35	| | | | | | | | $(W(r_2)\&H(r_2))$	27,19,ADJ
36	| | | | *show* $H(r_2)$	18,39,CONTRA
37	| | | | | $\neg H(r_2)$	ASSUME

```
38  | | | | | ¬N(r1)                                    7,37,MT
39  | | | | | *show (P(r1,r1)&B(r1))                  38,46,CONTRA
40  | | | |[ ] | *show P(r1,r1)                          26,41,CONTRA
41  | | | | | | | *show (P(r2,r1)&B(r2))               42,20,ADJ
42  |-| | | | | | | *show B(r2)                         44,45,CONTRA
43  | | | | | | | | | ¬B(r2)                            ASSUME
44  | | | | | | | | | ¬(W(r2)&H(r2))                    22,43,MT
45  | | | | | | | | | (W(r2)&H(r2))                     27,19,ADJ
46  |-| | | | | N(r1)                                    14,40,MP
```

real	user	system	
0.29	0.26	0.03	(sec.)

lines discarded: 33
303.85 lines/second (user time)

THE EXPLANATION IN TOP-DOWN FASHION CUT OFF AT LEVEL 4

To show that $(\forall x_0)((\forall y_0)(P(y_0,x_0)\rightarrow N(y_0))\rightarrow B(x_0))$ is true we first must prove $((\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))\rightarrow B(r_1))$ and then since r_1 is an arbitrarily chosen constant it can be replaced with a variable. We want to prove $((\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))\rightarrow B(r_1))$. To do this we shall assume $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$ and then show that $B(r_1)$ can then be derived. To prove $B(r_1)$ is true we will assume that it is false and then generate a contradiction from this assumption. We can get $(W(r_1)\&H(r_1))$ by proving both $H(r_1)$ and $W(r_1)$ then form a conjunction of the two. It is possible to prove that $H(r_1)$ is true. We will leave out the details of proving that $W(r_1)$ is true. We proved $(W(r_1)\&H(r_1))$ because $H(r_1)$ and $W(r_1)$ were shown to be true. Our assumption $\neg B(r_1)$ is false since we have a contradiction $(\neg(W(r_1)\&H(r_1)))$ and $((W(r_1)\&H(r_1)))$, therefore we get $B(r_1)$. We can now conclude $((\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))\rightarrow B(r_1))$ since $B(r_1)$ was derived after assuming $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$. We now have $(\forall x_0)((\forall y_0)(P(y_0,x_0)\rightarrow N(y_0))\rightarrow B(x_0))$ since we can replace the arbitrarily chosen constant r_1 in $((\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))\rightarrow B(r_1))$ by a variable.

real	user	system	
0.02	0.01	0.01	(sec.)

THE EXPLANATION IN BOTTOM-UP FASHION CUT OFF AT LEVEL 5

The main goal is to prove $(\forall x_0)((\forall y_0)(P(y_0,x_0)\rightarrow N(y_0))\rightarrow B(x_0))$. Suppose $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$. We know $(\forall x_0)((W(x_0)\&H(x_0))\rightarrow B(x_0))$ is a premise. We let $x_0=r_1$ in $(\forall x_0)((W(x_0)\&H(x_0))\rightarrow B(x_0))$. $\neg B(r_1)$ is assumed. It follows from these two the consequent is false so the antecedent must be false so we get $\neg(W(r_1)\&H(r_1))$. The statement $(\forall x_0)(\forall y_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$ is known to be true. We get $(\forall y_0)((P(y_0,r_1)\&B(y_0))\rightarrow B(r_1))$ by substituting r_1 for x_0 in the statement $(\forall x_0)(\forall y_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$. We get $((P(r_1,r_1)\&B(r_1))\rightarrow B(r_1))$ by substituting r_1 for y_0 in the statement $(\forall y_0)((P(y_0,r_1)\&B(y_0))\rightarrow B(r_1))$. Following the above formula and the last assumption, we get $\neg(P(r_1,r_1)\&B(r_1))$ since it can't be true because we know the consequent is false. We can prove $(P(r_1,r_1)\&B(r_1))$ in some other steps. We have both true $((P(r_1,r_1)\&B(r_1)))$ and false $(\neg(P(r_1,r_1)\&B(r_1)))$ of the same formula, which is a contradiction. Therefore, we must repeal our assumption $\neg H(r_1)$ and we have proven $H(r_1)$. We can prove $(P(r_1,r_1)\&B(r_1))$ in some other steps. We have generated a contradiction $(\neg(P(r_1,r_1)\&B(r_1)))$ and $((P(r_1,r_1)\&B(r_1)))$ and therefore we can conclude that $W(r_1)$. $H(r_1)$ was proved before. And it follows from these two we get $(W(r_1)\&H(r_1))$ by joining them with a conjunction. We have derived a contradiction $(\neg(W(r_1)\&H(r_1)))$ and $((W(r_1)\&H(r_1)))$ and so our assumption $\neg B(r_1)$ must be wrong and we get $B(r_1)$ as a result. We now have $((\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))\rightarrow B(r_1))$ because we assumed $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$ and then derived that $B(r_1)$ followed from that assumption. And following this, we can generalize to $(\forall x_0)((\forall y_0)(P(y_0,x_0)\rightarrow N(y_0))\rightarrow B(x_0))$ since r_1 was an arbitrary constant.

real	user	system	
0.04	0.02	0.02	(sec.)

THE EXPLANATION IN BOTTOM-UP FASHION OF THE SUBPROOF BEGINNING AT LINE 30

We want to prove $(P(r_1,r_1)\&B(r_1))$. The statement $(\forall x_0)(\forall y_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$ is known to be true. We set the variable x_0 to equal r_1 in $(\forall x_0)(\forall y_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$. $(\forall x_0)(\exists y_0)(W(y_0)\&P(y_0,x_0))$ is a fact. By replacing x_0 by r_1 in $(\forall x_0)(\exists y_0)(W(y_0)\&P(y_0,x_0))$ we get $(\exists y_0)(W(y_0)\&P(y_0,r_1))$. We get $(W(r_2)\&P(r_2,r_1))$ by replacing y_0 by a new constant r_2 in $(\exists y_0)(W(y_0)\&P(y_0,r_1))$. We substitute r_2 for y_0 in $(\forall y_0)((P(y_0,r_1)\&B(y_0))\rightarrow B(r_1))$. Suppose $\neg B(r_1)$. Using the last two formulas, since the consequent of a conditional is false the antecedent could not be true and we have $\neg(P(r_2,r_1)\&B(r_2))$. We know $(\forall x_0)((W(x_0)\&H(x_0))\rightarrow B(x_0))$ is a premise. We set the variable x_0 to equal r_2 in $(\forall x_0)((W(x_0)\&H(x_0))\rightarrow B(x_0))$. We assume $\neg B(r_2)$. From the previous two, we get $\neg(W(r_2)\&H(r_2))$ since it can't be true because we know the consequent is false. The statement $(\forall x_0)(N(x_0)\rightarrow H(x_0))$ is known to be true. We set the variable x_0 to equal r_2 in $(\forall x_0)(N(x_0)\rightarrow H(x_0))$. We assume $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$. We get $(P(r_2,r_1)\rightarrow N(r_2))$ by substituting r_2 for y_0 in the statement $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$. $P(r_2,r_1)$ was proved before. From the previous two, we derive that $N(r_2)$ because the antecedent being true implies that the consequent is also true. $(N(r_2)\rightarrow H(r_2))$ was proven earlier. From the previous two, since both the antecedent and the conditional are true we can derive the consequent of $H(r_2)$. As we saw earlier $W(r_2)$ is true. It follows from these two since they are both true or assumed to be true we join them and get the statement $(W(r_2)\&H(r_2))$. We have both true $((W(r_2)\&H(r_2)))$ and false $(\neg(W(r_2)\&H(r_2)))$ of the same formula, which is a contradiction. Therefore, we must repeal our assumption $\neg B(r_2)$ and we have proven $B(r_2)$. As we saw earlier $P(r_2,r_1)$ is true. And from the previous two, we can combine them into one formula to get $(P(r_2,r_1)\&B(r_2))$. We have generated a contradiction $(\neg(P(r_2,r_1)\&B(r_2)))$ and $((P(r_2,r_1)\&B(r_2)))$ and therefore we can conclude that $(P(r_1,r_1)\&B(r_1))$.

real	user	system	
0.05	0.03	0.02	(sec.)

THE EXPLANATION IN TOP-DOWN FASHION OF THE SUBPROOF BEGINNING AT LINE 39

To prove that $(P(r_1,r_1)\&B(r_1))$ is true we want to generate a contradiction. We can get $\neg N(r_1)$ if we can prove that both $(N(r_1)\rightarrow H(r_1))$ and $\neg H(r_1)$ are true. To prove that $(N(r_1)\rightarrow H(r_1))$ is true we have to show $(\forall x_0)(N(x_0)\rightarrow H(x_0))$ and then let $x_0=r_1$. It is known that $(\forall x_0)(N(x_0)\rightarrow H(x_0))$ is true. Now, let's suppose $\neg H(r_1)$. We can now get $\neg N(r_1)$ because the conditional $(N(r_1)\rightarrow H(r_1))$ was proved and we know $\neg H(r_1)$ therefore the antecedent couldn't be true. To prove $N(r_1)$ we want to prove that it can be derived from $P(r_1,r_1)$, and $(P(r_1,r_1)\rightarrow N(r_1))$ both being true. To get $(P(r_1,r_1)\rightarrow N(r_1))$ we must prove that $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$ is true and then substitute r_1 for y_0. We can also assume $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$. To prove that $P(r_1,r_1)$ is true we want to generate a contradiction. To get the statement $\neg(P(r_2,r_1)\&B(r_2))$ we must first show both $((P(r_2,r_1)\&B(r_2))\rightarrow B(r_1))$ and $\neg B(r_1)$. To show $(W(r_2)\&P(r_2,r_1))$ is true we have to prove $(\exists y_0)(W(y_0)\&P(y_0,r_1))$ and then we can instantiate the variable y_0 with the new constant r_2. We want to get $(\exists y_0)(W(y_0)\&P(y_0,r_1))$. To do this we prove $(\forall x_0)(\exists y_0)(W(y_0)\&P(y_0,x_0))$ and then substitute r_1 for x_0. We know $(\forall x_0)(\exists y_0)(W(y_0)\&P(y_0,x_0))$ is a premise. To get $((P(r_2,r_1)\&B(r_2))\rightarrow B(r_1))$ we must prove that $(\forall y_0)((P(y_0,r_1)\&B(y_0))\rightarrow B(r_1))$ is true and then substitute r_2 for y_0. To prove that $(\forall y_0)((P(y_0,r_1)\&B(y_0))\rightarrow B(r_1))$ is true we have to show $(\forall x_0)(\forall y_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$ and then let $x_0=r_1$. The statement $(\forall x_0)(\forall y_0)((P(y_0,x_0)\&B(y_0))\rightarrow B(x_0))$ is known to be true. $\neg B(r_1)$ is assumed. We can now conclude $\neg(P(r_2,r_1)\&B(r_2))$ because $\neg B(r_1)$ is assumed and $((P(r_2,r_1)\&B(r_2))\rightarrow B(r_1))$ was shown, so, the antecedent of the conditional could not be true. We can get $(P(r_2,r_1)\&B(r_2))$ by proving both $B(r_2)$ and $P(r_2,r_1)$ then form a conjunction of the two. We want to prove $B(r_2)$. To do this we will generate a contradiction when we assume it is false. To get $(W(r_2)\&H(r_2))$ we show $H(r_2)$ and $W(r_2)$ are true and then combine them in a

conjunction. To prove $H(r_2)$ we want to prove that it can be derived from $N(r_2)$, and $(N(r_2)\rightarrow H(r_2))$ both being true. To get $(N(r_2)\rightarrow H(r_2))$ we must prove that $(\forall x_0)(N(x_0)\rightarrow H(x_0))$ is true and then substitute r_2 for x_0. $(\forall x_0)(N(x_0)\rightarrow H(x_0))$, was previously given as a premise. To prove $N(r_2)$ we want to prove that it can be derived from $P(r_2,r_1)$, and $(P(r_2,r_1)\rightarrow N(r_2))$ both being true. We want to get $(P(r_2,r_1)\rightarrow N(r_2))$. To do this we prove $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$ and then substitute r_2 for y_0. We assumed $(\forall y_0)(P(y_0,r_1)\rightarrow N(y_0))$ earlier. We previously proved $(W(r_2)\&P(r_2,r_1))$. Since we proved $P(r_2,r_1)$, and $(P(r_2,r_1)\rightarrow N(r_2))$ we can get $N(r_2)$ because the first part of the conditional is true therefore the second part is also true. Since we proved $N(r_2)$, and $(N(r_2)\rightarrow H(r_2))$ we can get $H(r_2)$ because the first part of the conditional is true therefore the second part is also true. We proved $(W(r_2)\&P(r_2,r_1))$ earlier in the proof. We now get $(W(r_2)\&H(r_2))$ because we proved $H(r_2)$ and $W(r_2)$ were both true. We now have $B(r_2)$ because when we assumed that it was false we got a contradiction $(\neg(W(r_2)\&H(r_2)))$ and $((W(r_2)\&H(r_2)))$. $(W(r_2)\&P(r_2,r_1))$ was proved before. We proved $(P(r_2,r_1)\&B(r_2))$ because $B(r_2)$ and $P(r_2,r_1)$ were shown to be true. Since we have generated a contradiction $(\neg(P(r_2,r_1)\&B(r_2)))$ and $((P(r_2,r_1)\&B(r_2)))$ we can conclude $P(r_1,r_1)$. Since we proved $P(r_1,r_1)$, and $(P(r_1,r_1)\rightarrow N(r_1))$ we can get $N(r_1)$ because the first part of the conditional is true therefore the second part is also true. Since we have generated a contradiction $(\neg N(r_1))$ and $(N(r_1))$ we can conclude $(P(r_1,r_1)\&B(r_1))$.

real	user	system	
0.06	0.03	0.03	(sec.)

5. Future Developments and Conclusions

5.1 Areas for Future Work

There are several things that could be done as future work with respect to this thesis. The types of continuing work falls into two distinct areas. The first being experimentation with the types of explanations that EXPLAIN can produce. The second area of future research is in the further development of the program itself.

THINKER can prove a wide variety of proofs, most of which have specific attributes that distinguish one type of proof from another. It would be beneficial to be able to examine many different proofs along with several of their explanations to try to find explanation types that fit well with proof types. This type of cognitive testing can also be used to determine what is a good explanation and why is it better for a particular proof or subproof. For example, the bottom-up method seems to explain short straight line proofs (those with little depth) well. Where as top-down produces more understandable explanations for deeply embedded proofs. It may also become apparent that certain inference rules are explained best in a certain way. For example, it seems natural to explain Modus Ponens in a bottom-up fashion, rather than in a top-down manner. (A top-down manner would be like saying "We wish to prove q, so let's look for a formula p and try to prove $(p \rightarrow q)$ if we succeed, then do MP". A bottom-up explanation would be closer to how people reason: "We have p and also $(p \rightarrow q)$, so we infer q by MP".) Are there other inference rules that are best explained top-down or are there combinations of rules which should be explained one way or the other? If a match between explanation types and proofs can be made EXPLAIN could be improved to take advantage of this and produce

the better explanations automatically.

The further development of the program is the process of making EXPLAIN generate different types of explanations. The most obvious area for this expansion would be in dimension one that was explained earlier. The focus of the current work is on the static explanation of the proof as it stands. The opposite end of this dimension would be the dynamic recounting of the proof generation process. This would explain the strategy behind the construction of the proof with relevant information on the setting of subgoals and so forth. One way to produce this type of explanation would be to have THINKER generate a trace of the functions that it entered to produce the proof. This would give some information about what decisions were made to generate the proof without having to produce the explanation while the proof is actually being generated. This information could be used to generate the dynamic type of explanation after the proof was done.

The theorem prover THINKER has been expanded to be able to handle modal logic proofs as well as simple proofs. The explanation program EXPLAIN could also be expanded in this area to provide explanations for these types of proofs. This would entail creating text generating functions for the specific inference rules that deal with modal logic.

The type of explanations that are generated for any particular proof are guided solely by the user. An improvement to the system would be to have some type of user modeling. This way the system could anticipate the needs of the user and produce the explanations that would fit them best. Possible models could range from a novice in logic who would need complete predicate explanations, to an expert who may require just the inference rules and a high level type explanation. User modeling would relieve the user of having to find the correct type of explanation by themselves.

One final use of the system would be to explain what a given, individual formulas really says. For example, we might wish to explain what certain "axioms of rationality" [Alchourrón85] *really* say (and not merely how their authors interpret them). Since EXPLAIN can already explain a formula when the explanation is done to the level of the predicates or the connectives, this feature of the entire system can be put to use to explain formulas in isolation. This would be a simple extension where EXPLAIN is used without a proof being done.

5.2 Conclusions

The explanation facility EXPLAIN was written as an extension of the THINKER theorem prover that was implemented in "C" and runs on a SUN SPARC station. The time taken for the explanation of a proof to be produced is relatively short, as seen by the times at the end of the explanations in chapter 4. A statement of the algorithm employed to produce the explanations that appear in chapter 4 follows:

Main Explanation Function

1 Set parameters (from command line)

2 If predicate explanation READ lexical file

3 Remove unused lines from the linked list of the proof

4 If bottom-up explanation call pick_rule_BU with starting line number

5 If top-down explanation call pick_rule_TD with starting line number

pick_rule_BU

1 If the current line is the first line state it as the main goal

2 If current line's level is the current switching level call pick_rule_TD with current line and set the current switching level to the next level where switching is to take place

3 If current line's level is the cut-off level explain it as provable

4 If justification 1 exists call pick_rule_BU with justification 1

5 If justification 2 exists call pick_rule_BU with justification 2

6 Call line to explain current line's inference rule.

7 Set current line as explained at time X

pick_rule_TD

1 If current line's level is the current switching level call pick_rule_BU with current line and set the current switching level to the next level where switching is to take place

2 If current line's level is the cut-off level explain it as provable

3 Call line to explain current line's inference rule.

4 Set current line as explained at time X

BU inference rule functions

1 Produce the linking phrase (depending on when the justifications were explained)

2 Produce the explanations of the formulas to be used

3 Pick an explanation phrase for the inference rule

TD inference rule functions

1 If the line has been explained previously restate the line and return

2 Produce the explanations of the formulas to be used

3 State the current line as the goal

4 State the subgoal to be proved

5 Call pick_rule_TD with the justification lines

6 (optional) Explain how the justifications are combined to get the goal

Formula Explanation

1 If to the level of the inference rules return as is

2 If to the level of the connectives

2.1 Call function to explain the main connective (if it exists)

2.2 Explain the left subformula (if it exists)

2.3 Explain the right subformula (if it exists)

2.4 Pick a connective phrase to combine left and right explanations

2.5 If no main connective return the predicate as is

3 If to the level of the predicates

3.1 Call function to explain the main connective (if it exists)

3.2 If Universal or Existential quantifier

3.2.1 Make an entry in the constant list for the variable with a suitable phrase such as "everybody" or "someone"

3.2.2 Explain the right subformula

3.3 Explain the left subformula (if it exists)

3.4 Explain the right subformula (if it exists)

3.5 Pick a connective phrase to combine left and right explanations

3.6 If no connective call predicate explanation

Predicate explanation

1 If one argument

1.1 Should the negative phrase be used? (set by ¬ connective explanation)

1.2 Can a pronoun be used?

1.3 Is the phrase to be plural or singular?

2 If 2 arguments

2.1 Get the object phrase

2.2 Get the subject phrase

2.3 Check if self referral (i.e.; Fred shaved himself)

2.4 Positive or negative?

2.5 Any pronouns?

2.6 Singular or plural?

2.7 Active or passive? (randomly chosen)

5.3 Portability

The current implementation is not adaptable to any other automated theorem provers since EXPLAIN begins with the final internal form left by THINKER. To adapt this system to another theorem prover some preprocessing would have to be done to get the information in the correct form. To produce the explanations EXPLAIN has individual functions for the specific inference rules found in THINKER. To adapt EXPLAIN to other systems the inference rule explanation functions would have to be rewritten to match those appearing in that system, and to find some substitutes in that system that are equivalent to all the information that THINKER puts into a single line of a proof.

Bibliography

[Alchourrón85] C.E. Alchourrón, P. Gärdenfors and D. Makinson. "On the logic of theory change: partial meet contraction and revision functions", *Journal of Symbolic Logic* 50, pp. 510-530, 1985.

[Bibel79] W. Bibel, "Syntax-directed, semantic-supported program synthesis", *Proceedings of the 4th Workshop on Automated Deduction* , Austin(Texas), 1979, pp.140-147.

[Boyer84a] R. Boyer and J. Moore. "A mechanical proof of the unsolvability of the halting problem", *Journal of the ACM* 31, pp. 441-458, 1984.

[Boyer84b] R. Boyer and J. Moore. "Proof checking the RSA public key encryption algorithm", *American Mathematical Monthly* 91, pp. 181-189, 1984.

[Chester76] Daniel Chester, "The Translation of Formal Proofs into English", *Artificial Intelligence* 7 (1976), 261-278.

[Chisholm85] G. Chisholm, J. Kljaich, B. Smith, and A. Wojcik. "Preliminary report on the formal analysis of the Draper FTP Hardware and Software Using ITP", Technical Report MCS-TM-59, Mathematics and Computer Science Division, Argonne National Labratory, Argonne, Il September 1985.

[Felty88] Amy Felty and Greg Hager. "Explaining Model Logic Proofs" In *Proceedings of the 1988 IEEE International Conference on Systems, Man, and Cybernetics* , Beijing and Shenpang, China Vol 1 pp.177-180.

[Good82] D. Good. "The proof of a distributed system in Gypsy", Technical Report 30, Institute for Computing Science, University of Texas, Austin, TX, September 1982.

[Huang89a] Xiaorong Huang, "Proof Transformation Towards Human Reasoning Style", *Proceedings of the 13th German Workshop on Artificial Intelligence* , Ed. D. Metzing, Springer-Verlag, 1989, pp. 37-42.

[Huang89b] Xiaorong Huang, "A Human Oriented Proof Presentation Model", Technical Report SEKI SR-89-11, Kaiserslautern University, 1989.

[Huang90] Xiaorong Huang, "Reference Choices in Mathematical Proofs", *Proceedings of the 9th European Conference on Artificial Intelligence* , Pitman Publishing, 1990, pp. 720-725.

[Hunt86] W. Hunt. "The mechanical verification of a microprocessor", In *From H.D.L. Descriptions to Guaranteed Correct Circuit Designs* , North-Holland, New York, 1986.

[Lusk82a] E. Lusk, W. McCune, and R. Overbeek. "Logic Machine Architecture: kernal functions", in *Proceedings of the Sixth International Conference on Automated Deduction, Springer-Verlag Lecture Notes in Computer Science* , Vol. 183 (Edited by D.

Loveland) Springer-Verlag, New York, pp. 70-84, 1982.

[Lusk82b] E. Lusk, W. McCune, and R. Overbeek. "Logic Machine Architecture: inference mechanisms", in *Proceedings of the Sixth International Conference on Automated Deduction, Springer-Verlag Lecture Notes in Computer Science* , Vol. 183 (Edited by D. Loveland) Springer-Verlag, New York, pp. 85-108, 1982.

[McDonald85] David D. McDonald, "Natural language generation as a computational problem: an introduction." In *Computational models of Discourse* (Edited by M. Brady and R. Berwick) MIT Press Cambridge, Mass, 1985.

[Murray82] N. Murray, "Completely non-clausal Theorem Proving", *Artificial Intelligence* 18 (1982), 67-86.

[Winker79] S. Winker and L. Wos. "Automated generation of models and counterexamples and its application to open questions in ternary Boolean algebra", pp. 251-256 in *Proceedings of the Eighth International Symposium on Multiple-Valued Logic* , IEEE and ACM, Rosemont, IL, 1978.

[Winker81] S. Winker, L. Wos, and E. Lusk. "Semigroups, anitautomorphisms, and involutions: a computer solution to an open problem, I", *Mathematics of Computation* 37, pp. 533-545, 1981.

[Wojciechowski83] W. Wojciechowski and A. Wojcik. "Automated design of multiple-valued logic circuits by automatic theorem proving techniques", *IEEE Transactions on Computers* C-32, pp. 785-798, 1983.

[Wos84b] L. Wos, S. Winker, R. Veroff, B. Smith, and L. Henschen. "A new use of an automated reasoning assistant: open questions in equivalential calculus and the study of infinite domains", *Artificial Intelligence* 22, pp. 303-356, 1984.